AD-A089 424   GEORGE WASHINGTON UNIV WASHINGTON D C SCHOOL OF ENGI--ETC F/G 9/2
PERFORMANCE OPTIMIZATION OF THE PASCAL TO DEC VAX 11/780 MICROC--ETC(U)
AUG 80 R E MERWIN   DASG60-80-C-0006

UNCLASSIFIED   NL

1 OF 1
AD
A089424

END
DATE
FILMED
10-80
DTIC

THE
GEORGE
WASHINGTON
UNIVERSITY

STUDENTS FACULTY STUDY R
ESEARCH DEVELOPMENT FUT
URE CAREER CREATIVITY CC
MMUNITY LEADERSHIP TECH
NOLOGY FRONTI ESIGN
ENGINEERING APP ENC
GEORGE WASHI NIV

SCHOOL OF ENGINEERING
AND APPLIED SCIENCE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER -	2. GOVT ACCESSION NO. AD-A089 424	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Performance Optimization of the PASCAL to DEC VAX 11/780 Microcode Compiler.		5. TYPE OF REPORT & PERIOD COVERED 16 Nov. '79 to 16 Aug. '80
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) R.E. Merwin		8. CONTRACT OR GRANT NUMBER(s) DASG60-80-C-0006
9. PERFORMING ORGANIZATION NAME AND ADDRESS The George Washington University Washington, D.C. 20052		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE 18 Aug. '80
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclass.
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

SEP 3 4 1980

D

18. SUPPLEMENTARY NOTES

C

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Compiler, High level language, Microprogram, Quadruple, PASCAL, VAX 11/780

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Previous research activity at The George Washington University has demonstrated the feasibility of designing a compiler which accepts a high level (programming) language (HLL) and produces microcode for a computer with a horizontal control word format. On going research activity described herein is demonstrating the feasibility of designing a compiler with PASCAL as an input HLL that can generate microcode for the DEC VAX 11/780 computer which is installed in the BMDATC Distributed Data Processing (DDP) Test Bed. The effort described has as its goal the installation of this compiler at the Test Bed and to interface

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

it with the "User Microprogramming" support software provided by DEC. Additional tasks include optimizing the performance of the compiler and conducting a study of the interface between compiler generated microcode and CPU architectures.

(11) 18 Aug

(1) Performance Optimization of the
PASCAL to DEC VAX 11/780
Microcode Compiler.

(9) Final Report. 26 Nov
~~18 August 1980~~

The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

(10) R. E./Merwin

Sponsored by

The Ballistic Missile Defense Advanced Technology Center

(15) Contract No. DASG60-80-C-0006

School of Engineering and Applied Science

The George Washington University

Washington, D.C. 20052

ABSTRACT

Previous research activity at The George Washington University has demonstrated the feasibility of designing a compiler which accepts a high level (programming) language (HLL) and produces microcode for a computer with a horizontal control word format. On going research activity described herein is demonstrating the feasibility of designing a compiler with PASCAL as an input HLL that can generate microcode for the DEC VAX 11/780 computer which is installed in the BMDATC Distributed Data Processing (DDP) Test Bed. The effort described has as its goal the installation of this compiler at the Test Bed and to interface it with the "User Microprogramming" support software provided by DEC. Additional tasks include optimizing the performance of the compiler and conducting a study of the interface between compiler generated microcode and CPU architectures.

TABLE OF CONTENTS

INTRODUCTION

The use of microprogramming, especially for high usage segments of computer programs, as a way to enhance hardware performance has been demonstrated many times (1,2). Since the advent of minicomputers, circa 1970, the manufacturers of these systems have offered their customers a "user" microprogramming option. This consists of a small writable control storage (WCS) and micro assemblers and simulators to support generation of microprograms. There hasn't been a wide acceptance of this "user" microprogramming option by the purchasers of minicomputers primarily because generation of microprograms is an error prone and costly task. The basic support offered the "user" microprogrammer has simply been inadequate to overcome the difficulties of preparing and using microcode.

The purpose of the research described in this report is to develop additional tools to support the "user" microprogrammer. The use of high level languages (HLLs) for expressing computer programs are taken for granted by conventional software programmers and the use of this technique to generate microcode appears promising. Establishing feasibility that HLL compilers can be developed to produce efficient microcode is the main objective of this research activity. Efficient microcode in this context means microcoded routines which significantly enhances hardware performance on critical often used software segments hereafter referred to as kernals.

The research described in this report has been underway for about two years. In the beginning two compilers were available. One produced microcode from the MPL HLL for the INTERDATA Mod 3 and the other generated quadruples from the PLM HLL (3,4) . The INTERDATA Mod 3 was designed with a vertical control word format consisting of 16 bits and four modes. Generating microcode for this machine is in many ways similar to conventional assembly language programming. As was to be

expected, the compiler designed to produce INTERDATA Mod 3 microcode was quite efficient. The big challenge was to produce microcode via a compiler for a computer with a horizontal control word format. A horizontal control word may be from 50 to 100 bits wide and have from 20-40 control fields.

The first research effort in this activity consisted of using the PLM to quadruple compiler to generate microcode for the DEC PDP 11/45 machine (4). A quadruple to 11/45 microcode translator was developed and performance measured. The results were encouraging which lead to a much more challenging research effort (5) to change both the input to the compiler and to generate microcode for the DEC VAX 11/780 machine which offered a "user" microprogramming option. The input to the compiler was changed from PLM to PASCAL and the output again went via the intermediate quadruple format directly to microcode format of the VAX machine which has a control word consisting of 96 bits with nearly 30 control fields. Unfortunately, at the time of this research effort DEC hadn't released the "user" microprogramming support manuals and this greatly hindered our ability to determine if the microcode generated was correct and to measure its performance.

At the inception of the research effort of this present contract in November 1979, it was decided to transfer the compilers which produce microcode for the VAX machine to be operable on the BMDATC Test Bed VAX systems. A companion ARO contract (6) was established to transport the PASCAL to VAX microcode compiler from the IBM 370 so that it would execute on the VAX system. To do this, the XCOM compiler, which is part of the XPL Translator Writing System, was modified to produce quadruples as an output instead of 370 machine language.

An additional program was written to generate VAX MACRO language statements from the quadruples. This research activity has been slowed by many unusual events including a two month shut down of the Computer Center at GWU along with the necessity to replace the programmers working on the compiler twice. As a result this program fell behind schedule and wasn't able to support some aspects of this research effort.

Another major problem was the long delay in release of the VAX "user" microporgramming support documentation. This material wasn't received until April of '80 which hampered our ability to verify the correctness of the compiled VAX microcode. The combined effect of the two problem areas was to lead to our concentration on the quadruple to VAX microcode portion of the compiler. The PASCAL to quadruple compiler developed under the previous research effort was essentially assumed to work "as is" and the performance optimization and measurement activity concentrated on the quadruple to microcode translator. Fortunately, this approach worked quite well and good estimated performance measurements were obtained. Confirmation of these performance measurements on VAX systems at the BMDATC Test Bed are now underway.

There are four objectives for this research activity. The first was to complete the transfer of the PASCAL to VAX 11/780 microcode compiler to be operational on the BMDATC Test Bed. Achievement of this objective was contingent on the ARO companion effort noted above which has fallen behind schedule. Accordingly, the accomplishment of this objective has also fallen behind schedule. Rapid progress is being made and an operational PASCAL to VAX microcode compiler should be available at the BMDATC Test Bed in the fall of '80. The second objective was to measure the performance of the PASCAL to VAX microcode compiler.

This objective has been largely met with the exception of confirming estimated execution times with actual timing runs on the VAX systems at the BMDATC Test Bed. An attempt to do this failed because of systems software problems within the VAX VMS support software. The third objective was to optimize the PASCAL to VAX microcode compiler performance. Again this activity was concentrated on the quadruple to VAX microcode translator and good results were obtained. The main point here was to transfer data requests to the VAX internal registers to avoid the overhead of main storage references. The fourth objective was to study the interface between the compiler and the hardware microcode. This was done in conjunction with objective 3 and a better understanding of this interface was developed, however, much remains to be done in this research area.

The results obtained from this research effort are mainly to demonstrate that a HLL compiler can produce efficient microcode for the VAX machine. Specific results to be presented later indicate that compiler produced microcode is nearly comparable to microcode produced by hand. This should certainly be the case for the "occasional"microprogrammer who wouldn't develop the skill level that could be obtained by a full-time microprogrammer. Secondly the optimization techniques incorporated in the quadruple to VAX microcode translator work quite well and appear to be applicable to producing microcode for a wide variety of host machine hardwares. Finally a better understanding of the compiler to microcode generation process will lead to the specification of host machines which are better suited for interfacing to high level languages in general.

In view of the results obtained to date in this research activity to produce compilers from HLL's to microcode, it can be concluded the feasibility has been established and progress is being made in reduction-to-practice of this technique.

Adding this tool to the available microprogramming support should make "user" microprogramming a much more worthwhile option for purchasers of hardware with this feature. With the availability of cheaper hardware and the ever increasing cost of software, the use of microprogramming to enhance system flexibility appears to be an attractive advantage. This research activity will lead to better support tools for microprogrammers and ultimately to a wider use of this technique to improve both hardware performance and flexibility.

The final report includes six sections and four appendices. After this introductory section, there follows a section on research objectives and one on research accomplishments. A fourth section deals with compiler performance followed by a section on the HLL compiler hardware interface. The report concludes with a section containing summary comments on the significance of this research and suggestions for follow-on activity.

RESEARCH OBJECTIVES

Before beginning a detailed discussion of the objectives of this research effort, a few preliminaries are in order. The initial activity was to demonstrate the feasibility of producing with a compiler microcode for a computer employing a horizontal control word format. This was demonstrated (4) using the DEC PDP 11/45 as a host machine which has a 56 bit control word with nearly 20 control fields. This research revealed the importance of the compiler hardware interface and especially the necessity of taking advantage of available registers in the host machine to minimize main storage references. The next effort was an extension of the feasibility study to change the simple PLM source language for the compiler to PASCAL and to replace the DEC 11/45 by the VAX 11/780 as the host machine. The VAX 11/780 is a much more complex host machine than the 11/45 and utilizes a 96 bit control word format. Finally a companion effort was initiated to transport the PASCAL to VAX 11/780 microcode compiler to the VAX systems within the BMDATC Test Bed.

The first objective of this research contract was intended to be supportive of the other three and as a follow-on to the ARO contract objectives. By transferring the PASCAL to VAX microcode compiler to be operational at the BMDATC Test Bed, it was hoped that other BMDATC contractors would be encouraged to take advantage of the "user" microprogramming capability of the VAX hardware to enhance their system performance. To support this goal it was necessary to make the PASCAL to VAX microcode compiler available on the BMDATC Test Bed as well as provide support documentation as to how these facilities are to be used. As an initial step in this direction the PASCAL compiler had to be well documented and a "users" manual is required. In addition, the compiler must be made resident on the VAX systems at the BMDATC Test Bed and local systems programming personnel had to be familiarized with how to use this facility.

The second objective was to evaluate the performance of the PASCAL to VAX 11/780 microcode compiler. This would proceed through several phases starting with the relatively unoptimized version available at the inception of this research activity and moving on to the optimized versions of the compiler. The performance of compiler generated microcode would be measured against hand generated microcode and also against versions of the test programs coded in other high level languages available on the VAX system, i.e. mainly FORTRAN IV. Terms of measurement would be in running time and where possible in terms of control storage space utilization.

Objective three would be corollary with objective two in that optimization of the compiler would be carried out to provide better performance against hand coded microprograms and also in terms of running time against versions of the test programs coded in other HLL's resident on the VAX system. This objective has two aspects, namely optimization of the PASCAL to quadruple compiler and optimization of translation of quadruples to microcode. The latter aspect proved to be especially significant in generation of microcode for the PDP 11/45. Fairly standard optimization techniques will be considered in achieving this objective for the PASCAL to quadruple compiler. Register allocation algorithmns will be used to the maximum extent possible to optimize the quadruple to VAX microcode translator to improve run time performance and minimize the amount of control store usage.

The fourth objective is to study the interface between the PASCAL to VAX 11/780 microcode compiler and the CPU architecture. While this interface physically is at the microcode level, factors such as internal register availability, HLL features, and the choice of an intermediate representation all play

a role in determining total system performance. The experience gained in generating microcode for the PDP 11/45 from the PLM HLL demonstrated the importance of this objective. In generating test cases, it was discovered that for all but one of the test cases, Fibonacci Series, it wasn't possible to efficiently hand microprogram them because of the shortage of internal registers in the PDP 11/45 CPU. From the extent to which the goals of this objective are reached, we should be better able to determine architectural specifications of host machines to enhance their microprogrammability and ultimately their performance in a wide range of applications.

The four objectives for this research effort which are described above span a wide range of issues in the design of compilers to produce microcode. As in any research effort all objectives aren't equally achieved and as activity progresses new directions may appear and are pursued. Another factor is the research environment which can often lead to decisions to pursue different approaches to avoid obstacles that impede progress in the initially choosen directions. This contract activity encountered many problems and directions were changed to maximize the accomplishments as will be described in the next section.

PROJECT ACCOMPLISHMENTS

A number of accomplishments have been achieved as a result of the research activity into the design of compilers to convert computational algorithms expressed in a HLL into microcode. Some of the high lights include a high degree of optimization of the efficiency and performance of the microcode produced by the compiler which is executable on the VAX 11/780 CPU. Another significant accomplishment is the greater understanding of the interface between compilers which generate microcode and CPU archituectures. As design techniques for microprogrammed CPU's are improved, the interface between the software and firmware and these architectures becomes a more critical technical consideration. While many of the research objectives were met, not all were and in some cases a redirection was required to compensate for circumstances arising during the research activity.

Two events beyond our control contributed to the degree of accomplishment of the research objectives. The first was the delay until April '80 of the receipt of the complete DEC support documentation on how to microprogram the VAX CPU. Some preliminary and partial information became available in January '80 but lacking the complete documentation adversely affected the quadruple to microcode translator development. The second problem was the shut down of the computer facilities at GWU which were required to support this research. This led to a late start in achieving objective 1 to get the PASCAL compiler operational on the VAX System. A third problem, characteristic of a graduate school research environment, is the high rate of turnover of research staff. This research project was especially hard hit by the necessity of replacing two lead compiler programmers along with the unavoidable training period for replacements.

The accomplishments for each objective will be discussed below. It should be emphasized that this is an ongoing research effort and objectives not entirely achieved at the time this report is being written will be accomplished in the near future. The status of accomplishment for each objective will include an estimated completion date where appropriate.

Progress on objective one to make the PASCAL to VAX 11/780 microcode compiler operable on the BMDATC Test Bed is behind schedule by about two to three months for the reasons noted above. Work on interfacing this compiler with the DEC "User" Microprogramming support documentation is underway. A preliminary user's microprogramming manual has been prepared and is shown at appendix B. Of particular importance is the design of invoking programs which provide an interface between the "user" microprogram and the VMS operating system. An overview block diagram of the invoking procedure is shown in figure 1 and a sample invoking program is shown in figure 2. Another example of "user" documentation intended for operational personnel at the BMDATC Test Bed is shown at appendix A. It is a documented program written in the XPL language which converts quadruples to the VAX system MACRO language along with a variation of this translator which is still in development produces microcode for the VAX 11/780 from quadruples. In summary the status of objective one is that it is only partially complete and behind schedule. While some of the check out of the compiler can be carried out on the VAX 11/780 at GWU, a major effort will also be required at the Test Bed to bring it into operational status there.

Objective two is to evaluate the performance of the PASCAL to VAX 11/780 microcode compiler in terms of operational performance and microcode storage

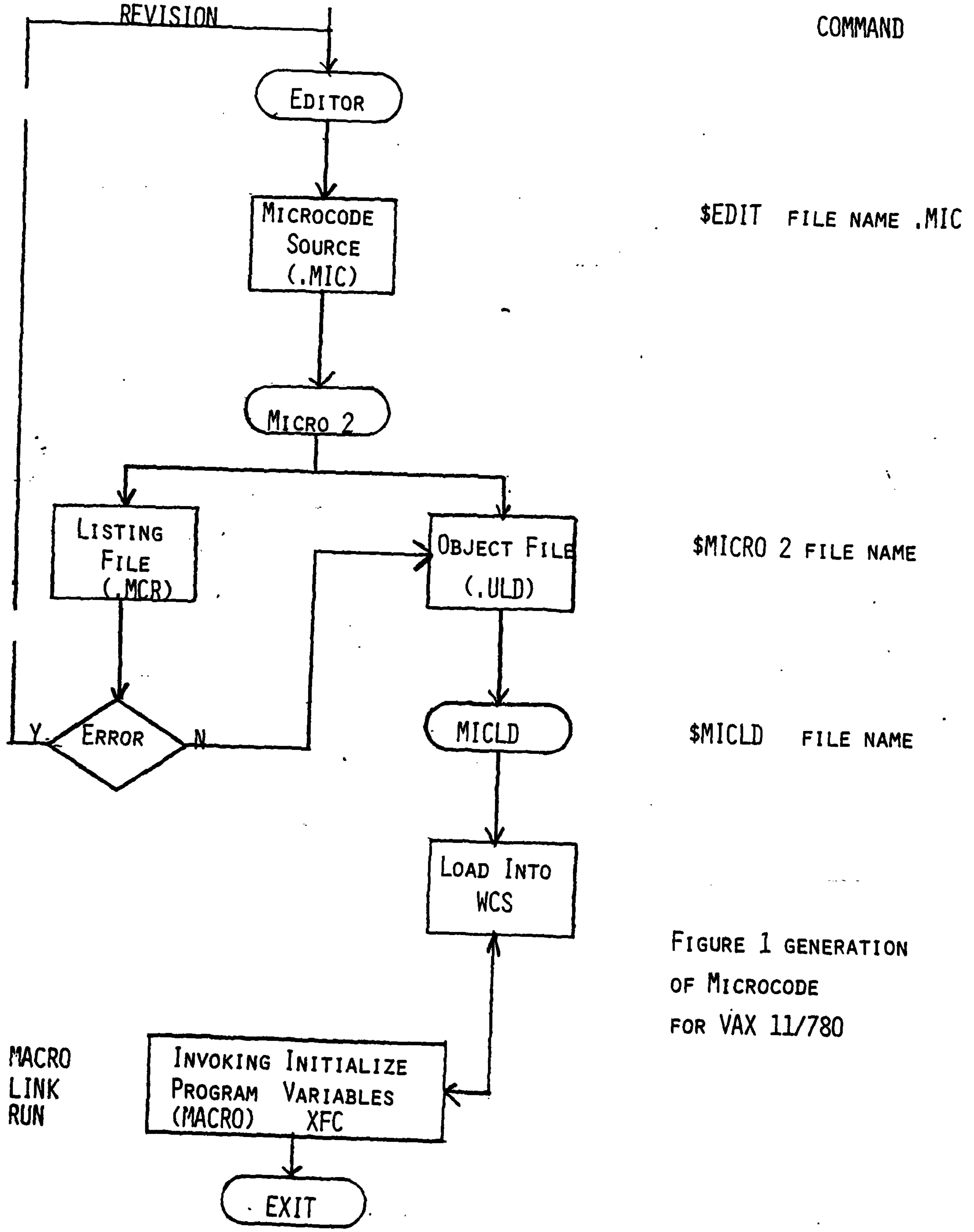


FIGURE 1 GENERATION OF MICROCODE FOR VAX 11/780

```
  TYPE  SORTTEST.MAR
.TITLE          SORTTEST
.ENTRY          SORTTEST,^M<>
$CMKRNL_S       ROUTIN=SAVEC    ;SAVE AND CHANGE XFC VECTOR IN SCB.
$CREATE         FAB=OUTFAB
$CONNECT        RAB=OUTRAB
$ASCTIM_S       ,TIMBUF=ATIMENOW,,
MOVC3           #32,ATIMENOW,TIMOUT
$PUT            RAB=OUTRAB
                MOVAL   AR,R0   ;STARTING ADDR OF ARRAY
;----------------;
XFC
;----------------;
$CMKRNL_S       ROUTIN=RSVEC
$EXIT_S
$ASCTIM_S       ,TIMBUF=ATIMENOW,,
MOVC3           #32,ATIMENOW,TIMOUT
$PUT            RAB=OUTRAB
$CLOSE          FAB=OUTFAB
;----------------;
SAVEC:          .WORD   0
                MOVL    SCB$AL_BASE+20,RIVEC
                MOVL    #2,SCB$AL_BASE+20
                RET
;----------------;
RSVEC:          .WORD   0
                MOVL    RIVEC,SCB$AL_BASE+20
                RET
;----------------;
                .PSECT  VECTOR,LONG
RIVEC:          .LONG   0
                .LONG   0
;----------------;
OUTFAB:         $FAB    FNM=TIMBF,RFM=FIX,MRS=TIMSIZ,RAT=CR
OUTRAB:         $RAB    FAB=OUTFAB,RBF=TIMOUT,RSZ=TIMSIZ
TIMOUT:         .BLKB   32
                TIMSIZ=.-TIMOUT
ATIMENOW:       .LONG   20$-10$
                .LONG 10$
10$:            .BLKB   32
20$:            .BLKB   0
AR:             .BLKL   255
                .END    SORTTEST
$
```

Figure 2 Microcode Invoking Program in VAX 11/780 MACRO Language

requirements. After getting off to a slow start due to lack of availability of the DEC support documentation for "user" microprogramming, considerable progress has been made in evaluating the compiler performance which looks quite acceptable. One missing detail for this objective is to conduct timing runs of the microcode produced both by hand and by compiler in the VAX systems installed at the Test Bed. An attempt was made to do this but failed due to problems with the VAX system software supporting the timing procedure. The test approach adopted for this objective was to initially code the four test programs in four ways. These are to code the test program in: DEC FORTRAN IV, PASCAL, microcode derived from quadruples, and hand coded microcode. Consideration was given to also coding the test programs in the VAX MACRO language but this was dropped when it became apparent that there would be little improvement in this representation over the MACRO version generated by the FORTRAN IV compiler. Timing tests were made on the FORTRAN IV version of the test programs and estimated execution times were generated for the quadruple and hand coded microcode versions. It is these latter versions that we were unable to get actual running times on the Test Bed VAX systems. Because of the importance of the accomplishments for objective 2, it will be covered in more detail in section four dealing with the compiler performance.

Accomplishments for objective three are corollary to those for objective 2. Optimization of the PASCAL to VAX 11/780 microcode compiler is decomposed into two parts. The first part deals with the PASCAL to quadruple compiler. Because of the problems noted above for objective one, it wasn't possible to make an attempt to significantly improve this compiler performance in a general way. It is also our observation from earlier research experience that significant gains

in microcode operational performance are unlikely to be due to the design of the compiler. Instead in accomplishing objective three all the stress was placed on optimizing the performance of the quadruple to VAX 11/780 microcode translator. Here significant gains in performance were obtained by taking advantage of allocating data to registers and combining quadruples into one microinstruction were possible. This required definition of special microcoded MACROS which is the interface to the "user" microprogrammer supported by the DEC microprogramming documentation. While all aspects of optimizing the generation of microcode from quadruples haven't been investigated, the performance of the microcode generated from quadruples compared to hand coded microcode looks very good. Again because of the significance of our accomplishments for this objective they will be dis-dussed in more detail in section four below.

The study of the HLL compiler CPU interface is an out growth of earlier observations of the critical nature of this boundary between software, firmware and hardware. Even more interesting is the trend, especially apparent in the VAX machine, for CPU architectures to directly reflect HLL representations. Examples include the support of procedures, subroutine calls, stacks, and a host of other HLL features at the machine language level. This leads to direct interpretation of these HLL features at the microcode level and eliminates the necessity of encoding these features in conventional machine language, e.g. the DEC PDP 11 assembly language. This introduction of HLL concepts directly into the machine language of the CPU tends to diminish the advantages of microprogramming computational algorithms to gain run time performance. As will be noted below this is born out by our performance evaluation showing only about 3 to 4 to one advantage

of hand microcoded algorithms over the FORTRAN IV equivalent. This shows the significance of this trend in CPU architecture design. While performance advantages may not be so great in using directly microprogrammed kernals in the VAX system, the flexibility of microprogramming can still offer lots of advantages. To investigate this further requires selection of algorithms which will achieve maximum benefit from microprogramming and evaluate the effectiveness of this representation as opposed to representations generated at the FORTRAN IV or MACRO language level. Again because of the importance of this objective, it is discussed in more detail in section five.

Considerable progress has been made on the objectives originally stated for this research effort in spite of the many obstacles encountered. Much remains to be done and the proposed on-going research addresses many of these issues. Most important, a good foundation is being established utilizing "state-of-the-art" hardware to develop tools to assist in the generation of microcode. With some estimates of the cost of generating a line of microcode running as high as $1000.00, any improvement in the productivity of microprogrammers can bring about significant savings. In spite of the trend of CPU manufacturers to incorporate HLL features directly in hardware, this flexibility of being able to directly microprogram critical software kernals makes the development of tools to assist the microprogrammer a high priority effort. As more is learned about the compilation procedures for microcode, better advantage can be taken of available hardware performance. It is toward this objective that the long range goals of this research effort are directed.

COMPILER PERFORMANCE

HLL Compiler performance can be measured in many ways. These include: impact on programmer productivity, readability and documentation of source code, computational thruput of the object code produced by the compiler, and amount of storage space required for a compiled program. From these fundamental measures other criteria can be derived including compiler efficiency (7) which is a comparison of computational thruput and storage requirements of an algorithm coded in a HLL and assembly language. To optimize the performance of a compiler, it is common practice to try to reduce the computational time, commonly referred to as "run time" of the generated object code. Optimization of the run time of the object code often requires additions to the time it takes to compile a program. If compile time also is an important factor, then tradeoffs may be required between object code optimization and compiler run time.

In this research project microcode execution time was the goal of the optimization effort. Compile time wasn't considered nor were control storage requirements. As noted before, optimization activity was directed at the quadruple to microcode translator. The main factors to be considered were the allocation of registers to the variables being modified at any particular segment of the program, enablement of branch operations, and accessing data arrays. By careful design of the translator program, we were able to generate microcode which was comparable in run time performance to hand generated microcode.

To test the compiler two approaches were used. The first was to make timing estimates by counting the number of microinstructions that would be executed to carry out the computations required for the test cases. The execution time of each microinstruction, which is typically 200 nonoseconds, were used as factors to

estimate test program run times. The second approach is to create a microprogram load module and actually execute it on the VAX systems in the BMDATC Test Bed. Because of the granularity (10 milliseconds) of the elapsed time measurement facility on the VAX system, it is necessary to run the test case many times in order to be able to make a run time estimate. The control of the number of test cycles to be run must be included within the microcode and this introduces a bias in the run time estimate. There is an additional timing overhead introduced by the invoking program as it passes control to the microprogram. This is a one time bias which doesn't seriously affect the accuracy of the run time tests. In order to reduce the impact of these test errors, the number of test run cycles is varied to obtain an average run time for each test case.

To illustrate the compilation procedure an example is shown in figure 3. This shows a sort routine which sorts an array of numbers into ascending order. This program has a best and worst case test requirement. The best case is when all the elements to be sorted are already in ascending order and the worst case occurs when they are initially all in descending order. It is evident from this figure as to where some overhead is inherent in microcode generated by a compiler. Do loops require indices to be specified and comparisons must be made on iteration variables. For branches and if-then-else constructs it is necessary to compute a predicate before determining whether to go to the branch address or not. In spite of these overhead aspects of compiled microcode, the run time performance of microcode generated from quadruples is nearly comparable to hand generated microcode as is shown below.

The quadruple to microcode translator was designed to produce as efficient microcode as possible. The design approach was to first microcode the test problems by hand and compare it to the microcode derived by translation from quad-

```
        .TITLE "QBUBBLE";
        .REGION /1400,17FF;
;BASIC PROGRAM FLOW:--;
;INITIALIZE FILE TO BE SORTED--;
;INITIALIZE SORT PROGRAM VARIABLES IS & K--;
;SET UP OUTER DO LOOP FOR SORT BASED ON INDEX K--;
;SET UP INNER DO LOOP FOR SORT BASED ON INDEX I FOR FILE AR--;
;ON COMPLETION OF SORT PASS ON INNER LOOP AND IS=1--;
;DECREMENT K AND REPEAT INNER LOOP--;
;WHEN K=0 OR IS=0 AT END OF INNER PASS EXIT PROGRAM--;
;INPUT
;       R0 AR MEMORY BASE ADDR.
;LABELS
;       ADDR2 START OF SORT K(OUTER) LOOP
;       ADDR3 LAST STMT OF INTERCHANGE LOOP
;       ADDR4 START OF SORT FILE LOAD LOOP
;       ADDR5 INPUT TO SORT INDICE INITIALIZATION
;       ADDR6 BEGIN INTERCHANGE LOOP
;       ADDR7 LAST STMT OF SORT K(OUTER) LOOP
;       ADDR8 EXIT FROM SORT PROGRAM
;INTERNAL VARIABLES
;       I INDEX TO ARRAY AR STORED IN REG 1
;       IS INTERCHANGE SIGNAL STORED IN REG 2
;       ITEMP WORKING SPACE FOR INTERCHANGE PROG. IN REG 3
;       K OUTER SORT LOOP INDEX STORED IN REG 4
;INITIALIZATION OF ARRAY TO BE SORTED
;IT WILL CONSIST OF 255 ELEMENTS IN
;INVERSE ORDER
```

Figure 3 - Example of PASCAL To QUADRUPLE To VAX 11/780 MICRO 2 ASSEMBLY LANGUAGE COMPILATION PROCEDURE FOR SORT TEST CASE.

PASCAL SOURCE	OPER	OPND 1	OPND 2	OPND 3	VAX MICRO 2
	QUADRUPLES				
AR (1): = 255;	INDX	U . . . S0		T1[4]	---; Q_R[R0]; AR BASE ADDR TO Q REG ---; R[R5]_ALU,ALU_Q+K[.1]; AR(0)+1 TO REG 5
	MOV	# 255		@T1	---; D_K[.FF]; 255 TO D REG ---; VA_ALU,ALU_LA,LAB_R[R5]; ADDR AR(1) TO MA ---; CACHE_D[LONG]; 255 TO AR(1)
FOR I: = 2 to 255 DO;	MOV	# 2		U . . . S1	---; R[R1]_K[.2]; I=2
	ADDR 4				;------- ADDR4:
	GT	U . . . S1	# 255	T1	Q_K[.FF]; 255 TO Q REG ---; ALU_LA-Q,LAB_R[R1],CLK.UBCC; IS I > 255? ---; ALU?;
	BRT	ADDR 5	T1		=1010 J/ADDR5; COMPLETION OF SORT FILE LOA
AR (I-1)=AR(I)-1;	ADD	U . . . S0	U . . . S1	T1	---; LAB_R[R0]; ADDR OF AR TO AB LATCH ---; Q_R[R1]; PUT INDEX I FOR AR INTO Q REG ---; R[R5]_ALU,ALU_Q+LB; ADDR OF AR(I) TO R5
	SUB	@ T1	#1	T2	---; VA_ALU,ALU_LA,LAB_R[R5]; ADDR AR(I) TO MA ---; D[LONG]_CACHE; AR(I) TO D REG ---; R[R6]_ALU,ALU_D-K[.1]; AR(I)-1 TO REG 6

Figure 3-2

PASCAL SOURCE	QUADRUPLES OPER	OPND 1	OPND 2	OPND 3	VAX MICRO 2 / COMMENT
	SUB	T1	#1	T1	---; LAB_R[R5]; ADDR OF AR(I) TO AB LATCH ---; R[R5]_ALU,ALU_LA-K[.1]; ADDR OF AR(I-1) TO REG 5
	MOV	T2		@T1	---; D_R[R6]; AR(I)-1 TO D REG ---; VA_R[R5]; ADDR OF AR(I-1) TO MAR ---; CACHE_D[LONG]; AR(I)-1 TO STORAGE
) CONTINUE	ADD	U . . . S1	#1	S . . . 2	---; LAB_R[R1]; I IDEX TO AB LATCH ---; R[R1]_ALU,ALU_LA+K[.1],J/ADDR4; INCREMENT I
	ADDR 5				;------- ADDR5:
: = 0	MOV	#0		S . . . 2	R[R2]_K[ZERO]; SET VAR IS=0 INTERCHANGE SIGNAL
- 250	MOV	#255		S . . . 4	R[R4]_ALU,ALU_K[.FF]; SET VAR K=255 ;NUMBER OF ELEMENTS TO BE SORTED;
= K-1	ADDR 2				ADDR2:
	SUB	S . . . 4	#1	S . . . 4	LAB_R[R4]; K TO AB LATCH ---; ALU_LA-K[.1],R[R4]_ALU; K-I TO K
R I: = 1 To K DO;	MOV	#1		U . . . S1	---; R[R1]_K[.1]; SET I=1
	ADDR 6				ADDR6:
	GT	U . . . S4	U . . . S1	T	Q_R[R4]; K TO Q REG ---; ALU_LA-Q,LAB_R[R1],CLK,UBCC; IS K-I>0 ---; ALU7:
	BRT	ADDR 7	T		;--- =1010 J/ADDR7;

Figure 3-3

PASCAL SOURCE	QUADRUPLES				VAX MICRO 2	COMMENT
	OPER	OPND 1	OPND 2	OPND 3		
F AR(I) < AR(I + 1) Then Go to ADDR 3	ADD	U . . .SO	U . . .S1	T1	---; LAB_R[R0]; ---; Q_R[R1]; ---; R[R5]_ALU,ALU_Q+LB;	 BASE ADDR OF AR TO AB LATCH I TO Q REG AR BASE ADDR+1 TO R5
	ADD	#1	T1	T2	---; LAB_R[R5]; ---; ALU_LA+K[.1],R[R6]_ALU;	 AR(I) TO AB LATCH AR(I+1) TO REG 6
	LT	@T1	@T2	T	---; VA_R[R6]; ---; D[LONG]_CACHE; ---; Q_D; ---; VA_R[R5]; ---; D[LONG]_CACHE;	 ADDR OF AR(I+1) TO MAR AR(I+1) TO D REG AR(I+1) TO D REG ADDR OF AR(I) TO MAR AR(I) TO D REG
	BRT	ADDR 3			---; ALU_Q-D,CLK,UBCC; ---; ALUT; ---; J/ADDR3;	 AR(I+1)-AR(I) BR TO END OF EXCHANGE LOOP
: = 1	MOV	#1			---; R[R2]_K[.1];	 SET IS=1 EXCHANGE SWITCH
EMP: = AR(I)	ADD	U . . .SO	U . . .S1	T1	---; LAB_R[R0]; ---; Q_R[R1]; ---; R[R5]_ALU, ALU_Q+LB;	 AR BASE ADDR TO AB LATCH I TO Q REG AR(I) TO REG5
	MOV	@T1		U . . .S3	---; VA_R[R5]; ---; D[LONG]_CACHE; ---; R[R3]_D;	 AR(I) TO MAR AR(I) TO D REG AR(I) TO ITEMP (REG3)

Figure 3-4

PASCAL SOURCE	QUADRUPLES OPER	OPND 1	OPND 2	OPND 3	VAX MICRO 2	COMMENT
R (I): = AR(I + 1)	ADD	U . . .S0	U . . .S1	T1	---; LAB_R[R0];	AR BASE ADDR TO AB LATCH
					---; Q_R[R1];	I TO Q REG
					---; R[R5]_ALU,ALU_Q+LB;	AR(0)+I TO REG 5
	ADD	T1	#1	T2	---; LAB_R[R5];	AR(I) TO AB LATCH
					---; ALU_LA+K[.1],R[R6]_ALU;	AR(I+1) TO REG6
R (I + 1): = ITEMP	ADD	#1	T1	T2	---; LAB_R[R6];	ADDR OF AR(I+1) TO AB LATCH
					---; ALU_LA+K[.1], R[R6]_ALU;	AR(I+1) TO REG 6
	MOV	U . . .S3		@T3	---; VA_R[R6];	AR(I+1) TO MAR
					---; D_R[R3];	ITEMP TO D REG
					---; CACHE_D[LONG];	ITEMP TO AR(I+1)
	ADDR 3				ADDR3:	
ONTINUE DO	ADD	#1		U . . .S1	LAB_R[R1];	I TO AB LATCH
					---; R[R1]_ALU,ALU_LA+K[.1];	I=I+1
	BR	ADDR6			--; J/ADDR6;	RETURN TO SORT LOOP
	ADDR 7				ADDR7:	

Figure 3-5

PASCAL SOURCE	QUADRUPLES				VAX MICRO 2	MMENT
	OPER	OPND 1	OPND 2	OPND 3		
JFK>OTHEN Go to ADDR 2	GT	U . . . S4	#0	TO	LAB_R[R4]; K TO AB LATCH -; ALU_LA-K[ZERO],CLK,UBCC; IF K-O=O THEN BR TO ADDR ;K-O BR TO ADDR2 IF LESSTHAN; -; ALUT; IS THERE AN ALU OUTPUT	
	BRT	ADDR 2			=1010 J/ADDR2;	
END	END				PC_PC+1,CLR,IB,OPC,J/062; END OF QBUBBLE ---;	

Fioure 3-6

ruples. The differences were examined and the translator was modified to reduce this difference to the maximum extent possible. In some cases this involves changing the quadruple input from the PASCAL to quadruple compiler. Changes are being made to the compiler to produce the desired quadruples.

A final effort, which is currently in process, is to redefine the MICRO 2 MACROS to better reflect the translation of quadruples into microcode. The MICRO 2 MACROS are a set of micro assembly language routines which are used by DEC Engineers in the design of the microprograms to interpret the VAX machine instructions set. Several instances have been detected where new MICRO 2 MACRO definitions would offer still better hardware performance for microcode generated from quadruples.

A major consideration in the design of both the PASCAL to quadruple compiler and quadruple to microcode translator are precise definitions of the quadruple formats and specifications in terms of MICRO 2 MACROS of each quadruple type. The quadruple consists of an operation, two source operands, and a destination operand. Each operand can be expressed in terms of different data types. This gives an extremely wide range of quadruple formats and data definitions. Appendix C lists the basic quadruple formats along with a representative set of quadruple types. Note that for each operator there are a number of potential operand data types leading to a wide variety of quadruples.

Appendix D contains a table showing the MICRO 2 MACROS required to implement the various quadruple types. Again a given quadruple operator may have a number of MICRO 2 MACRO representations corresponding to various operand data

types. Finally in appendix E a couple representative microcoded test cases are shown. Some of these as indicated are derived from quadruple representations and some are representative of hand coded microcode.

The data obtained for the estimated run times is shown in table I. In general the microcode produced by the PASCAL compiler had a run time which averaged 45% longer than the run time of the equivalent hand generated microcode. The same test problems coded in DEC FORTRAN IV and actually timed on the VAX system had an average run time which was 312% longer than the hand generated microcode. The average speed advantage of the PASCAL to microcode compiler over the FORTRAN compiler is 2.22 to one. This result is a little surprising since in general going directly to microcode should produce a bigger speed advantage. Part of the reason is the inclusion of high level language instructions in the VAX 11/780 instruction set which facilitates the direct interpretation of a HLL on the VAX system. This issue will be more extensively discussed in the next section.

From the data in table I, we can conclude that the PASCAL to microcode compiler is an effective software development tool. It produces quite efficient microcode which is nearly as efficient as the computational performance of hand generated microcode. The optimization effort on the quadruple to microcode translator was quite successful as is evidenced by the efficiency of the microcode produced. Further effort should be expended on the quadruple to microcode translator to see if its performance can be improved still further.

Test Case / Compiler	Greatest Element (10)	Fibonacci (40)	Prime (250)	Bubble Sort (250)	
				Worst	Best
Hand Coded Microcode	28.19	64.06	1400	$.14\times10^6$	$.06\times10^6$
Pascal to Quad to Micro 2	37.09	100	1700	$.26\times10^6$	$.08\times10^6$
Fortran	90	200	5000	$.36\times10^6$	$.19\times10^6$

TABLE 1 PERFORMANCE COMPARISON

PASCAL To Microcode Compiler

ALL Times in microseconds

HIGH LEVEL LANGUAGE-HARDWARE INTERFACE

Interest in the interface between the HLL compiler and the CPU hardware developed during the first phase of this research activity (4) when it was demonstrated that microcode could be generated for CPU's with a horizontal control word format. Because of the inherent machine independence of most HLLs, it was necessary to introduce some intermediate representation which was also relatively machine independent but could be translated into efficient microcode. Quadruples were selected for this representation but other choices could be made, e.g. triples, polish notation, or tree structures. Many of these alternate choices imply the use of a push down stack to implement the translation into microcode.

The selection of quadruples as an intermediate representation was largely motivated by the availability of a compiler which produced quadruples from the PLM language. While quadruples are relatively machine independent, in many ways they are equivalent to machine language. Quadruples are directly interpreted in microcode just as machine instructions are. The basic difference is that machine instructions deal with specific entities within the CPU, e.g. registers, constant generators, shifters, and data formats. Quadruples on the other hand specify only operations and operand types and addresses. These must first be translated into a format more representative of the CPU architecture before being converted into microcode. The translation from quadruples to a similar but machine dependent format is the basic task of the quadruple to microcode translator. Interpretation of the machine dependent quadruple representation is relatively straight forward.

A significant difference between the VAX 11/780 and the PDP 11/45 CPUs is the inclusion of HLL constructs in the machine language of the VAX system. As

noted above this significantly improves the VAX 11/780 system performance in executing algorithms expressed in a HLL which can be translated directly into these machine language expressions. Another feature of the VAX 11/780 machine instruction set is its rich variety of operand formats. This permits literally thousands of different instructions to be generated from only a couple hundred basic types. On the one hand this presents a serious problem to the compiler designer who must attempt to use all the available CPU facilities. The price paid for all this flexibility is longer run time and storage space being required for the compilation procedure. On the other hand the run time performance of compiled machine code begins to approach that of hand generated microcode.

Tradeoffs must be made by the CPU designers to provide features which makes the hardware perform efficiently when most of the software to be executed is originally expressed in a HLL. One danger, of course, is to create a machine instruction set which is too biased towards a particular HLL. This doesn't appear to be the case for the VAX 11/780 but this will be better known when performance data becomes available for HLL compilers for languages other than FORTRAN IV i.e., PASCAL, ADA, and C language which produce object code for the VAX system become available.

Of particular significance to the generation of efficient microcode from quadruples is the availability of registers to the microprogrammer. The VAX 11/780 has seven general purpose registers available at both the machine instruction and microprogram level. In addition there are sixteen scratch pad registers available to the microprogrammer. This is a vast improvement over the registers available in the PDP 11/45 where there were only six general purpose registers available both to the machine language programmer and microprogrammer. Lack of registers made array access very difficult and inefficient in the PDP 11/45 and prevented efficient hand microcoding of most of the test cases used in this research effort. Array manipulations can be easily handled within the VAX CPU and this leads to

significant increases in run time performance of algorithms requiring array accesses.

For a given set of registers, the allocation of data to registers is the critical factor in achieving optimization of translating quadruples into microcode. The procedure is to scan through the quadruples and as operands involving data or addresses are identified, an assignment is made to an available register. If no register is available, then a deallocation algorithm must decide which data item should be removed from the registers. These functions are all carried out at compile time and don't impact the run time performance of the microprogram load module except to the extent that data allocation to registers requires shifting data back and forth between main storage and the CPU registers.

Another key factor in generating microcode is the available data paths within the CPU. Any operation on data for a particular microinstruction requires that the data flow within the CPU is on separate data busses. Determining whether this is the case is a very tedious and error prone procedure. Fortunately the MICRO 2 microcode assembler does a good job of detecting data path usage conflicts and provides error messages to the microprogrammer. The tendency in the design of early minicomputers was to minimize the number of available data paths which had a severe impact on CPU performance. The VAX 11/780 CPU includes enough data paths to mostly avoid having to generate additional microinstructions to resolve data path conflicts. Some improvements could be made in the VAX data path structure but in general, the available data paths are more than adequate to support the other internal processing functions in the CPU.

In summary then the main factors in the compiler to CPU interface are: available machine instructions which directly reflect HLL structure, the supply of registers to permit storage of most of the active computation variables, and provision of adequate data path capacity to avoid having to generate additional microinstructions to resolve data path conflicts. The VAX 11/780 addresses these issues to a much better degree than did the PDP 11/45. As a result the run time performance of HLL compiler generated machine language and microcode approach the performance obtainable from hand coded microcode. Based upon these observations, more study of the compiler CPU interface should be carried out to determine more optimum designs for both the hardware and the compilers.

CONCLUSION

The overall research objectives of this contract were met except where uncontrollable circumstances intervened. A design has been achieved of a PASCAL to VAX 11/780 microcode compiler which produces efficient microprograms and the basic problems of interfacing this compiler to the VAX operating system have been solved. Preliminary support documentation has been prepared to support the "user" microprogramming option for the VAX system by other users of the BMDATC Test Bed.

Because of the efforts to optimize the performance of the PASCAL to microcode compiler, a much better understanding of the interface between a HLL and the microcode control of the CPU was developed. In particular algorithms to achieve optimum use of the registers to store run time variables were of especial interest and importance. The understanding gained of the generation of microcode from compilers can lead to significant improvement in productivity of microprogrammers and expand the capability for the use of this performance enhancing facility.

A further outgrowth of this research effort is the building of a broad base for future research into the techniques of enhancing computer performance thru microprogramming. Of particular significance is the development of host machine register assignment routines and the invoking programs required to pass control to and from the "user" microprogram. With these and other accomplishments, the foundation has been laid for a broad based research effort to develop and perfect support tools for the microprogramming activity.

While much has been accomplished, future research is required to develop techniques for enhancing the flexibility of microcode compilers by demonstrating that they can be converted at the input HLL interface to accept new computer

languages and at the host machine interface to demonstrate they can produce microcode for new host machines. This presents a severe challenge to researchers in this field and much fundamental work remains to be accomplished. The payoff from this research can be the ability to make better use of microprogramming both to enhance the computational performance of a given host machine and make it more flexible with regard to implementation of special purpose computational algorithms.

The research activity described in this final report was participated in by several individuals at The George Washington University. In particular I want to acknowledge the efforts of Robert Jones who did this original layout of the PASCAL and revised XCOM compilers. Mohamoud Ketabchi and Abdol Chenari continued the compiler development and were later joined in this activity by Mr. Kwang. Mr. Teh-Hsin Yang carried out the compiler optimization and evaluation effort.

REFERENCES

1. Doucette, D.R., "Performance Enhancement by Special Instructions on the System/360, Models 40 and 50," Preprints of the Third ACM Workshop on Microprogramming, Buffalo, N.Y. Oct. 1970.

2. Pager, D., "Some Notes on Speeding up Certain Loops by Software, Firmware, and Hardware Means," IEEETC, C21, January '72, pp. 97-100.

3. Fodor, P.R., "Evaluation of Compiler Design and Performance for a Vertical Microprogrammed Machine," MSEE Thesis, The George Washington University, Washington, D.C. May 1976.

4. Merwin, R.E., "Development of Experimental Compilers to Generate Emulators for the BMD DDP Test Bed from High Level Languages," Final Report Contract #DASG60-78-C-0115, Sponsored by USA BMDATC, The George Washington University, 1 April '79.

5. "PASCAL to Microcode Compiler Development," Final Report Subcontract #H07868AF9S, Sponsored by TRW Inc., The George Washington University, Sept. '79.

6. Merwin, R.D., "Development of a Compiler to Generate Microcode for the VAX 11/780 from the PASCAL HLL," Final Report, Contract DAAG29-76-D-0100, Battelle Columbus Laboratories, Delivery Order No. 1339-02, Sept. 1980.

7. Henriksen, J.O. and R.E. Merwin, "Programming Language Efficiency in Real-Time Software Systems," AFIPS Proc. SJCC, 40(1972), pp. 155-161.

Appendix A

A1 Quadruple To VAX 11/780 MACRO Language Translator Showing Documentation in Terms of Comments for Each Procedure

A2 Quadruple To VAX 11/780 Microcode Translator with Example of Microcode Derived from Quadruples for Fibonacci Test Case.

X P L COMPILATION -- GEORGE WASHINGTON UNIVERSITY -- XCOM IIIA VERSION OF JANUARY 5, 1973. CLOCK TIME = 17:23:2.13.

TODAY IS SEPTEMBER 1, 1980. CLOCK TIME = 18:49:9.30.

```
| /*****************************************************************************/|  1286
| /*                          DECLARATION                                       */|  1286
| /*****************************************************************************/|  1286
|  /* THE QUAD BUFFERS */                                                          |  1286
|    DECLARE QBUFSIZE LITERALLY '50';                                             |  1286
|    DECLARE (QUADNAME,QOP1,QOP2,QOP3,QUADTYPE,QIDBUF)(QBUFSIZE) CHARACTER,      |  1286
|            QBRFLG(QBUFSIZE) BIT(1);                                             |  1286
|    /*                                                                           |  1286
|    QUAD RELATED VARIABLES                                                       |  1286
|                                                                     */          |  1286
|    DECLARE MAX#_OF_FBR LITERALLY '9';                                           |  1286
|    DECLARE QNUM FIXED,                                                          |  1286
|             (QUAD_GROUP,QUAD#_IN_GROUP,NVAL_OF_QID) BIT(8),                     |  1286
|             #OF_QUADS FIXED INITIAL(0),                                         |  1286
|             FBRDESQ(MAX#_OF_FBR) FIXED,                                         |  1286
|             FBRDESQ_IND BIT(8) INITIAL(0),                                      |  1286
|             SKIP_NEXT_QUAD BIT(1) INITIAL(0);                                   |  1286
|    /*                                                                           |  1286
|    THE VAX_ASSEMBLY BUFFERS                                                     |  1286
|                                                                      */         |  1286
|    DECLARE VBUFSIZE LITERALLY '80';                                             |  1286
|    DECLARE (VLABEL_FLD,VOPERAT_FLD,VOPRND_FLD) (VBUFSIZE) CHARACTER,           |  1286
|            VQNUM(VBUFSIZE) FIXED;                                               |  1286
|    /*                                                                           |  1286
|    VAX_ASSEMBLY RELATED VARIABLES                                               |  1286
|                                                                      */         |  1286
|    DECLARE #OF_GVINS FIXED,                                                     |  1286
|            (VOPERAT,VOPRND) CHARACTER,                                          |  1286
|            DECL# FIXED INITIAL(80);                                             |  1286
|    /*                                                                           |  1286
|    HOUSEKEEPING VARIABLES                                                       |  1286
|                                                                      */         |  1286
|    DECLARE MAX#_OF_SYMBS LITERALLY'20',                                         |  1286
|             DIGIT LITERALLY '0123456789';                                       |  1286
|    DECLARE (NAME,CTYPE)(MAX#_OF_SYMBS) CHARACTER,                               |  1286
|             (GEN_REG#,OP1CT,OP2CT,OP3CT,OPXCT,OPND) CHARACTER,                  |  1286
|             LABL# BIT(8) INITIAL(0),                                            |  1286
|             FLABL# BIT(8) INITIAL(0),                                           |  1286
|             INDX_SYMB(3) CHARACTER,INDX_REG CHARACTER,(NDXED_OP# BIT(8),        |  1286
|             AFT_INDX BIT(8) INITIAL(0),INDX# BIT(8) INITIAL(0),                 |  1286
|             #OF_DEFSYMB FIXED INITIAL(0);                                       |  1286
| /*****************************************************************************/|  1286
| /*                          PROCEDURE STOR_VAX_INS                            */|  1286
| /*                          LL---6                                            */|  1286
| /*                                                                            */|  1286
| /*   THIS PROCEDURE FORMATS EACH VAX MACRO LANGUAGE STATEMENT AFTER BEING     */|  1286
| /*CONVERTED FROM QUADS. THE PARAMETERS RECEIVED ARE OPERATGOR, OPERANDS,      */|  1286
| /*QUAD NUMBER AND NUMBER OF VAX INSTRUCTIONS TO BE GENERATED.                 */|  1286
| /*****************************************************************************/|  1286
| STOR_VAX_INS:PROCEDURE(VOPERAT,VOPRND,VQUAD#,VINSNUM);                          |  1286
|   DECLARE (VOPERAT,VOPRND) CHARACTER,(VQUAD#,VINSNUM) FIXED;                    |  1286 STOR_VAX_INS
|     VOPERAT_FLD(VINSNUM)=VOPERAT;                                               |  1294 STOR_VAX_INS
|     VOPRND_FLD(VINSNUM)=VOPRND;                                                 |  1310 STOR_VAX_INS
|       VQNUM(VINSNUM)=VQUAD#;                                                    |  1326 STOR_VAX_INS
|                                                                                 |  1342 STOR_VAX_INS
```

```
/*****************************************************************************/|
/*                        PROCEDURE SEARCH _TYPE                            */|  1360
/*                        LL---5                                            */|  1360
/*                                                                          */|  1360
/*   OPX IS THE OPERAND WHOSE TYPE IS TO BE DETERMINED.                    */|  1360
/*   THIS PROCEDURE DETERMINES THE DATA TYPE OF A PARTICULAR OPERAND       */|  1360
/*WHICH IS PASSED TO THE CALLING PROCEDURE. THIS TYPES ARE CONTAINED IN THE */|  1360
/*NAME & TYPE TABLE.                                                        */|  1360
/****************************************************************************/|  1360
SEARCH_TYPE:PROCEDURE(OPX) CHARACTER;                                         |  1360
  DECLARE OPX CHARACTER, I FIXED;                                             |  1360 SEARCH_TYPE
  /* RETURNS TYPE OF OPX FROM NAME & CTYPE TABLES */                          |  1368 SEARCH_TYPE
    DO I=0 TO #OF_DEFSYMB;                                                    |  1368 SEARCH_TYPE
      IF NAME(I)=OPX THEN RETURN CTYPE(I);                                    |  1402 SEARCH_TYPE
    END; /* DO I */                                                           |  1464 SEARCH_TYPE
END SEARCH_TYPE;                                                              |  1468 SEARCH_TYPE
/****************************************************************************/|  1474
/*                        PROCEDURE SRCH_QNUM                               */|  1474
/*                        LL---4                                            */|  1474
/*                                                                          */|  1474
/*   THIS PROCEDURE FINDS THE CORRESPONDING VAX INSTRUCTION TO WHICH       */|  1474
/*THE EXECUTION IS TO BRANCH. FIRST THE FLAGE 'LOCN' SET TO -1, THEN EACH  */|  1474
/*VAX INSTRUCTION IS SCANNED TO SEE IF THE QUAD#=VQNUM(I). IF IT IS, THEN  */|  1474
/*THE CORRESPONDING QUAD IS FOUND. THE FLAGE 'LOCN' IS SET TO '1'. IF IT IS */|  1474
/*NOT FOUND, 'LOCN' RETURNS THE VALUE '-1'.                                 */|  1474
/****************************************************************************/|  1474
SRCH_QNUM:PROCEDURE(QUAD#) FIXED;                                             |  1474
  DECLARE (LOW,HIGH,I,LOCN,QUAD#) FIXED;                                      |  1474 SRCH_QNUM
    /* RETURNS THE VAX INSTRUCTIONS NUM ER CORRESPONDING TO QUAD#.            |  1482 SRCH_QNUM
        IF QUAD# IS NOT THERE RETURNS -1 */                                   |  1482 SRCH_QNUM
     LOCN=-1;                                                                 |  1482 SRCH_QNUM
     LOW=1;                                                                   |  1490 SRCH_QNUM
     HIGH=#OF_GVINS;                                                          |  1498 SRCH_QNUM
     DO WHILE (HIGH>0)&(HIGH>LOW);                                            |  1506 SRCH_QNUM
       I=(HIGH+LOW)/2;                                                        |  1556 SRCH_QNUM
       IF QUAD#=VQNUM(I) THEN  /* FOUND */                                    |  1578 SRCH_QNUM C7 = 2.
         DO;                                                                  |  1606 SRCH_QNUM
         LOCN=I;                                                              |  1598 SRCH_QNUM
         IF QUAD#=VQNUM(I-1) THEN LOCN=I-1;                                   |  1606 SRCH_QNUM
         RETURN LOCN;                                                         |  1642 SRCH_QNUM
         END; /* OF IF_THEN DO */                                             |  1652 SRCH_QNUM
       ELSE DO;                                                               |  1652 SRCH_QNUM
         IF QUAD#<VQNUM(I-1) THEN HIGH=I-1;                                   |  1656 SRCH_QNUM
         ELSE LOW=I+1;                                                        |  1692 SRCH_QNUM
       END; /* OF ELSE DO */                                                  |  1708 SRCH_QNUM
     END; /* OF DO WHILE*/                                                    |  1708 SRCH_QNUM
     RETURN LOCN;                                                             |  1712 SRCH_QNUM
END SRCH_QNUM;                                                                |  1722 SRCH_QNUM
/****************************************************************************/|  1728
/*                        PROCEDURE SYMBTAYL                                */|  1728
/*                        LL---3                                            */|  1728
/*                                                                          */|  1728
/*   OPX IS THE CHARACTER STRING OF THE QUAD OPERAND                       */|  1728
/*   THE PROCEDURE IS TO SCAN THE FIRST TWO CHARACTERS OF THIS QUAD OPERAND */|  1728
/*TO FIND ANY NON-BLANK AND NON-ZERO CHARACTERS. THE PROCEDURE RETURNS ITS */|  1728
/*VARIABLE 'SYMBOL' WHICH CONTAINS THE FIRST NON ZERO CHARACTERS AND ANY   */|  1728
/*TRAILING NON ZERO CHARACTERS IN THIS OPERAND AND DELETES ALL INTERVENING */|  1728
/*ZEROS.                                                                    */|  1728
/****************************************************************************/|  1728
SYMBTAYL:PROCEDURE(OPX) CHARACTER;                                            |  1728
  DECLARE (OPX,SYMBOL) CHARACTER,(QOPLENG,I) FIXED;                           |  1728 SYMBTAYL
  QOPLENG=LENGTH(OPX);                                                        |  1736 SYMBTAYL
  SYMBOL='';                                                                  |  1758 SYMBTAYL
  IF SUBSTR(OPX,0,1)¬=' ' THEN SYMBOL=SUBSTR(OPX,0,1);                        |  1764 SYMBTAYL
    IF SUBSTR(OPX,1,1)¬='0' THEN SYMBOL=SYMBOL||SUBSTR(OPX,1,1);              |  1822 SYMBTAYL
```

```
       '=I+1;                                                                 |  1962 SYMBTAYL C8 = 240. C9 = 11.
    E   /* OF DO_WHILE */                                                     |  1974 SYMBTAYL
    IF (BYTE(OPX,10)=BYTE('0'))&(I=11) THEN SYMBOL=SYMBOL||'0'.               |  1978 SYMBTAYL
      ELSE SYMBOL=SYMBOL||SUBSTR(OPX,I,QOPLENG-I);                            |  2060 SYMBTAYL
    RETURN SYMBOL;                                                            |  2114 SYMBTAYL
  END SYMBTAYL;                                                               |  2124 SYMBTAYL
  /*************************************************************************/|  2130
  /*                         PROCEDURE BLANK                                */|  2130
  /*                         LL---2                                         */|  2130
  /*                                                                        */|  2130
  /*      THIS PROCEDURE ALIGNS THE OUTPUT STRING IN THE RIGHT POSITION BY  */|  2130
  /*INSERTING BLANKS SO THAT EACH FIELD IN THE INSTRUCTION STARTS FROM THE  */|  2130
  /*SAME COLUMN.                                                            */|  2130
  /*************************************************************************/|  2130
  BLANK:PROCEDURE(STRING,WIDTH) CHARACTER;                                    |  2130
    DECLARE(WIDTH,L) FIXED, STRING CHARACTER;                                 |  2130 BLANK
    /* RETURN THE "STRING" CONCATENATED WITH BLANKS TO MAKE THE               |  2138 BLANK
          LENGTH EQUAL TO "WIDTH" */                                          |  2138 BLANK
    L=LENGTH(STRING);                                                         |  2138 BLANK
    IF L>= WIDTH THEN RETURN STRING;                                          |  2160 BLANK
    ELSE RETURN STRING||SUBSTR('                 ',0,WIDTH-L);                |  2182 BLANK
  END BLANK;                                                                  |  2234 BLANK
  /*************************************************************************/|  2240
  /*                         CONVERT_TO_INTEG                               */|  2240
  /*                         LL---1                                         */|  2240
  /*                                                                        */|  2240
  /*    'S' IS A CHARACTER STRING TO BE CONVERTED INTO AN INTEGER.          */|  2240
  /*    THE PROCEDURE IS FIRST TO CHECK TO SEE IF THE LEADING BYTE IS '-'   */|  2240
  /*(MINUS SIGN). IF IT IS, CONVERT THE REMAINDER OF THE STRING INTO        */|  2240
  /*INTEGER. IF THE LEADING BYTE IS NOT '-', THEN, CONVERT THE WHOLE STRING.*/|  2240
  /*PASS THE RESULTING VALUE TO THE CALLING PROCEDURE.                      */|  2240
  /*************************************************************************/|  2240
  CONVERT_TO_INTEG:PROCEDURE(S) BIT(8);                                       |  2240
    DECLARE S CHARACTER, NEGATIVE BIT(1),(I,NVAL) BIT(8);                     |  2240 CONVERT_TO_INTEG
    /* CONVERTS CHARACTER STRING "S" TO AN INTER */                           |  2248 CONVERT_TO_INTEG
    NEGATIVE=BYTE(S)=BYTE('-');                                               |  2248 CONVERT_TO_INTEG
    NVAL=0;                                                                   |  2278 CONVERT_TO_INTEG C10 = 96.
    DO I=NEGATIVE TO LENGTH(S)-1;                                             |  2284 CONVERT_TO_INTEG
      NVAL=NVAL*10+BYTE(S,I)-BYTE('0');                                       |  2342 CONVERT_TO_INTEG
    END;                                                                      |  2376 CONVERT_TO_INTEG C11 = 10.
    IF NEGATIVE THEN RETURN -NVAL;                                            |  2380 CONVERT_TO_INTEG
    ELSE RETURN NVAL;                                                         |  2404 CONVERT_TO_INTEG
  END CONVERT_TO_INTEG;                                                       |  2420 CONVERT_TO_INTEG
  /*************************************************************************/|  2426
  /*                         PROCEDURE SRCH_INDX                            */|  2426
  /*                         L6                                             */|  2426
  /*                                                                        */|  2426
  /*    THIS PROCEDURE FINDS OUT WHETHER THE QUADRUPLE OPERATOR IS INDEX(62).*/| 2426
  /*IF IT IS INDEXED, GENERATE A INDEXED REGISTER. OTHERWISE RETURN THE LETTER */| 2426
  /*'?'                                                                     */|  2426
  /*************************************************************************/|  2426
  SRCH_INDX:PROCEDURE(OP_SYMB,INDXNUM) CHARACTER;                             |  2426
    DECLARE OP_SYMB CHARACTER,(FOUND,INDXNUM,I)BIT(8);                        |  2426 SRCH_INDX
    I,FOUND=0;                                                                |  2434 SRCH_INDX
    DO WHILE (I<=INDXNUM)&(FOUND=0);                                          |  2444 SRCH_INDX
      IF INDX_SYMB(I)=OP_SYMB THEN FOUND=1;                                   |  2502 SRCH_INDX
      I=I+1;                                                                  |  2554 SRCH_INDX
    END; /* OF DO WHILE */                                                    |  2568 SRCH_INDX
    IF I<=INDXNUM THEN RETURN 'R'||SUBSTR(DIGIT,I+5,1);                       |  2572 SRCH_INDX
      ELSE RETURN '?';                                                        |  2642 SRCH_INDX C12 = 123456789. C13 = 5.
  END SRCH_INDX;                                                              |  2656 SRCH_INDX
  /*************************************************************************/|  2662
  /*                         PROCEDURE IDENT_UTOPX                          */|  2662
  /*                         L5                                             */|  2662
```

```
/*   THIS PROCEDURE IS TO PROCESS 'U' TYPE OPERAND. FIRST THE PROCEDURE       */|  2662
/*S  _INDX IS CALLED TO SEE IF IT IS INDEXED (I.E. HAS A R'  STER               */|  2662
/*ASSIGNMENT). IF IT IS INDEXED, THEN THE INDX_OP IS ASSIGNE  TO THE OPND AND*/|  2662
/*'R' IS SET TO THE OPXCT. IF IT IS NOT, CALL PROCEDURE SEARCH_TYPE TO        */|  2662
/*EXAMINE THE DATA TYPE OF THE OPERAND AND IF THE FLAG AFT_INDX=1 AND         */|  2662
/*INDEXED_OP#=OP#, THEN GENERATE AN INDEXED REGISTER TO THE OPERAND AND RESET*/|  2662
/*THE FLAG TO ZERO.                                                           */|  2662
/*****************************************************************************/|  2662
IDENT_UTOPX:PROCEDURE(OP#);                                                     |  2662
  DECLARE INDX_OP CHARACTER, OP# BIT(8);                                        |  2662 IDENT_UTOPX
  IF BYTE (OPND)=BYTE('U') THEN                                                 |  2670 IDENT_UTOPX
    DO;                                                                         |  2696 IDENT_UTOPX C14 = 228.
    INDX_OP=SRCH_INDX(OPND,INDX#);                                              |  2688 IDENT_UTOPX
    IF INDX_OP¬='?' THEN                                                        |  2714 IDENT_UTOPX
      DO;                                                                       |  2756 IDENT_UTOPX
      OPND=INDX_OP;                                                             |  2748 IDENT_UTOPX
      OPXCT='R';                                                                |  2756 IDENT_UTOPX
       END; /* OF IF THEN DO */                                                 |  2764 IDENT_UTOPX
    ELSE DO;                                                                    |  2764 IDENT_UTOPX
      OPXCT=SEARCH_TYPE(OPND);                                                  |  2768 IDENT_UTOPX
      IF (AFT_INDX=1)&(INDXED_OP#=OP#) THEN                                     |  2784 IDENT_UTOPX
        DO;                                                                     |  2834 IDENT_UTOPX
        OPND=OPND||'('||INDX_REG||')';                                          |  2842 IDENT_UTOPX
        AFT_INDX=0;                                                             |  2898 IDENT_UTOPX
        INDX_REG='';                                                            |  2904 IDENT_UTOPX
        END; /* OF IF THEN DO */                                                |  2910 IDENT_UTOPX
    END; /* OF ELSE DO */                                                       |  2910 IDENT_UTOPX
  END; /* OF IF THEN DO */                                                      |  2910 IDENT_UTOPX
END IDENT_UTOPX;                                                                |  2910 IDENT_UTOPX
/*****************************************************************************/|  2916
   /*                    PROCEDURE IDEN_OPX                                   */|  2916
/*                       L4                                                   */|  2916
/*                                                                            */|  2916
/*   OPX# IS OPERAND NUMBER 1,2,3.                                            */|  2916
/*   TMP_OP(3) IS AN ARRAY OF 4 ELEMENTS WHICH ARE INITIALLY 'T0','T1','T2', */|  2916
/*   'T3'.                                                                    */|  2916
/*   THE SYMBTAYL PROCEDURE OPERATES ON EACH OPERAND FROM THE QUAD SPECIFIED */|  2916
/*'QNUM'. NUMBER OF OPX(1,2,3) SELECTS A PARTICULAR OPERAND STORED IN OPND.  */|  2916
/*THE THREE MAIN IF STATEMENTS DETERMINE WHETHER THE FIRST CHARACTER OF THE  */|  2916
/*OPERAND IS 'U','T' OR '#'. IF IT IS 'U', THEN PROCEDURE 'IDENT-UTOPX' IS   */|  2916
/*CALLED. IF IT IS 'T', OPND IS COMPARED TO TMP_OP TO SEE IF IT IS 'T0','T1',*/|  2916
/*'T2' OR 'T3'. IF IT IS, A REGISTER IS GENERATED BY USING FILE DIGIT AND THE*/|  2916
/*LETTER 'R' IS CONCATENATED WITH OPND AND OPXCT IS SET TO 'R'. IF THE FIRST */|  2916
/*CHARACTER IS '#', THEN OPXCT IS SET TO '#'.                                 */|  2916
/*****************************************************************************/|  2916
IDENT_OPX:PROCEDURE(OP#);                                                       |  2916
  DECLARE (OP#,I) BIT(8),TMP_OP(3) CHARACTER INITIAL('T0','T1','T2','T3');      |  2916 IDENT_OPX
  /*  TAKES OP# AND FINDS ITS TYPE FROM NAME & CTYPE TABLES OR                  |  2924 IDENT_OPX
        ATRIBUTES TYPE IF IT IS # OR T TYPE */                                  |  2924 IDENT_OPX
  DO CASE OP#;                                                                  |  2924 IDENT_OPX
    OPND=SYMBTAYL(QOP1(QNUM)); /* OPX IS 1ST OPERAND */                         |  2946 IDENT_OPX
    OPND=SYMBTAYL(QOP2(QNUM)); /* OPX IS 2ND OPERAND */                         |  2970 IDENT_OPX CASE 0.
    OPND=SYMBTAYL(QOP3(QNUM)); /* OPX IS 3ND OPERAND */                         |  2998 IDENT_OPX CASE 1.
  END;  /* OF CASE STATEMENT */                                                 |  3026 IDENT_OPX CASE 2.
    IF BYTE(OPND)=BYTE('U') THEN                                                |  3030 IDENT_OPX
      CALL IDENT_UTOPX(OP#);                                                    |  3056 IDENT_OPX
    IF BYTE(OPND)=BYTE('T') THEN /* OPERAND IS AT TYPE SYMBOL */                |  3062 IDENT_OPX
      DO;                                                                       |  3098 IDENT_OPX C15 = 227.
      DO I=0 TO 3;                                                              |  3080 IDENT_OPX
        IF OPND=TMP_OP(I) THEN GEN_REG#=SUBSTR(DIGIT,I+3,1);                    |  3108 IDENT_OPX C16 = 3.
        END; /* OF DO */                                                        |  3186 IDENT_OPX
      OPND='R'||GEN_REG#;                                                       |  3190 IDENT_OPX
      OPXCT='R';                                                                |  3214 IDENT_OPX
```

```
END IDENT_OPX;                                                                       3248 IDENT_OPX
/************************************************************     ****************/  3254
/*                        PROCEDURE CONV_DATA                                    */  3254
/*                        L4                                                     */  3254
/*                                                                               */  3254
/*   THIS PROCEDURE CONVERTS DATA TYPE OF FIRST OPERAND OF A QUADRUPLE INTO      */  3254
/*THE DATA TYPE OF THE 3RD OPERAND.                                              */  3254
/*********************************************************************************/  3254
CONV_DATA:PROCEDURE(ORG_DES,ORGOP,GRG#);                                             3254
  DECLARE (ORG_DES,ORGOP,GRG#) CHARACTER;                                            3254 CONV_DATA
  VOPERAT='CVT'||ORG_DES;                                                            3262 CONV_DATA
  VOPRND=ORGOP||','||'R'||GRG#;                                                      3286 CONV_DATA
  CALL STOR_VAX_INS(VOPERAT,VOPRND,QNUM,#OF_GVINS);                                  3342 CONV_DATA
END CONV_DATA;                                                                       3378 CONV_DATA
/*********************************************************************************/  3384
/*                        PROCEDURE GLABL                                        */  3384
/*                        L4--5,1--1                                             */  3384
/*                                                                               */  3384
/*   THIS PROCEDURE MAY GENERATE A LABLE AS OUTPUT NLABL. FIRST, CHECK TO SEE*/  3384
/*THE LEADING CHARACTER IS 'U', IF IT IS 'U', SYMBTAYL IS CALLED AND THE         */  3384
/*LETTER 'L' IS PUT IN FRONT OF 'NLABL'. IF IT IS NOT, CHECK TO SEE IF THE       */  3384
/*DESTINATION QUAD HAS BEEN PROCESSED. IF IT HAS BEEN PROCESSED, THE             */  3384
/*SRCH_QNUM PROCEDURE IS CALLED. IF VINSQ#=-1, THEN PRINT OUT 'REFERENCED        */  3384
/*QUAD IS NOT FOUND'. IF IT IS FOUND, CHECK TO SEE IF THE INSTRUCTION ALREADY*/  3384
/*HAS A LABLE. IF IT DOESN'T HAVE A LABLE, GENERATE A LABLE. IF IT HAS A         */  3384
/*LABLE, THEN GENERATE A LABLE OPERAND FOR THE VAX BRANCH INSTRUCTION. IF THE*/  3384
/*DESTINATION QUAD HAS NOT BEEN PROCESSED, SET THE FLAG 'QBRFLG' TO 1,           */  3384
/*GENERATE A LABLE. PASS THE 'NLABL' TO THE CALLING PROCEDURE.                   */  3384
/*********************************************************************************/  3384
GLABL:PROCEDURE(QNUM_OP) CHARACTER;                                                  3384
  DECLARE (QUADN,VINSQ#) FIXED, (QNUM_OP,NLABL) CHARACTER;                           3384 GLABL
  IF BYTE(QNUM_OP,1)=BYTE('U') THEN                                                  3392 GLABL
  DO;                                                                                3420 GLABL
  NLABL=SYMBTAYL(QNUM_OP);                                                           3412 GLABL
  NLABL='L'||NLABL;                                                                  3428 GLABL
  END;  /* OF IF THEN DO */                                                          3452 GLABL
  ELSE DO;                                                                           3452 GLABL
    QUADN=CONVERT_TO_INTEG(QNUM_OP);                                                 3456 GLABL
    IF QUADN<=QNUM THEN                                                              3472 GLABL
      DO;  /* DESTINATION QUAD HAS BEEN PROCESSED */                                 3492 GLABL
    VINSQ#=SRCH_QNUM(QUADN);  /* FIND THE INSTRUCTION CORRESPONDING TO QUADN */      3484 GLABL
      IF VINSQ#=-1 THEN OUTPUT='REFERENCED QUAD NOT FOUND';                          3500 GLABL
    /* CHECK TO SEE IF THE INSTRUCGTION ALREADY HAS A LABLE  */                      3532 GLABL
    IF BYTE(VLABEL_FLD(VINSQ#))¬=BYTE('L') THEN                                      3532 GLABL
      DO;  /* DOES NOT HAVE LABEL, PUT ONE */                                        3566 GLABL C18 = 211.
        NLABL='LABL'||LABL#;                                                         3558 GLABL
        VLABEL_FLD(VINSQ#)=NLABL||':';                                               3592 GLABL
        LABL#=LABL#+1;                                                               3632 GLABL
      END;  /* OF IF THEN DO */                                                      3646 GLABL
    ELSE NLABL=SUBSTR(VLABEL_FLD(VINSQ#),0,5);  /* ALREADY HAS LABEL */              3646 GLABL
      END;  /* OF IF THEN DO */                                                      3674 GLABL C19 = 67108864.
    ELSE DO;                                                                         3674 GLABL
     IF QIDBUF(QUADN)='24' THEN                                                      3678 GLABL
        DO;                                                                          3728 GLABL
        NLABL=SYMBTAYL(QOP1(QUADN));                                                 3720 GLABL
        NLABL='L'||NLABL;                                                            3744 GLABL
        END;  /* OF IF THEN DO */                                                    3768 GLABL
        ELSE DO;                                                                     3768 GLABL
        QBRFLG(QUADN)=1;                                                             3772 GLABL
        FBRDESQ(FBRDESQ_IND)=QUADN;                                                  3784 GLABL
        NLABL='LBLF'||SUBSTR(DIGIT,FBRDESQ_IND,1);                                   3802 GLABL
        FBRDESQ_IND=FBRDESQ_IND+1;                                                   3848 GLABL
        END;  /* OF ELSE DO */                                                       3862 GLABL
```

```
    RETURN DLABL;
 END GLABL;                                                                     |  3872 GLABL
/** ***********************************************   ***************/|  3878
/*                       PRODECURE CHOOS_BRTYP                             */|  3878
/*                       L4---1--1                                         */|  3878
/*                                                                         */|  3878
/*   THIS PROCEDURE IS TO CHOOSE A CORRESPONDING BRANCH INSTRUCTION BASED ON */|  3878
/*THE VALUE OF 'BRTYP'. THE PROCEDURE GENERATES A LABLE AND SETS THE FLAG   */|  3878
/*SKIP_TO_NEXT TO 1. VAX INSTRUCTION IS SELECTED BASED ON TRUE COMPARISON, OR*/|  3878
/*ELSE, THE BRANCH IS DONE ON FALSE COMPARISON.                             */|  3878
/***************************************************************************/|  3878
CHOOS_BRTYP:PROCEDURE(COMP_KIND,BRTYP);                                        |  3878
  DECLARE (COMP_KIND,BRTYP) BIT(8), DLABL CHARACTER;                           |  3878 CHOOS_BRTYP
    DLABL=GLABL(QOP1(QNUM+1));                                                 |  3886 CHOOS_BRTYP
    SKIP_NEXT_QUAD=1;                                                          |  3914 CHOOS_BRTYP
    IF BRTYP=1 THEN  /* BRANCH IS DONE ON TRUE COMPARISON */                   |  3922 CHOOS_BRTYP
      DO CASE COMP_KIND;                                                       |  3944 CHOOS_BRTYP
        CALL STOR_VAX_INS('BGTR',DLABL,QNUM+1,#OF_GVINS);                      |  3958 CHOOS_BRTYP
        CALL STOR_VAX_INS('BLEQ',DLABL,QNUM+1,#OF_GVINS);                      |  3998 CHOOS_BRTYP CASE 0.
        CALL STOR_VAX_INS('BNEQ',DLABL,QNUM+1,#OF_GVINS);                      |  4042 CHOOS_BRTYP CASE 1.
        CALL STOR_VAX_INS('BLSS',DLABL,QNUM+1,#OF_GVINS);                      |  4086 CHOOS_BRTYP CASE 2.
        CALL STOR_VAX_INS('BGTR',DLABL,QNUM+1,#OF_GVINS);                      |  4130 CHOOS_BRTYP CASE 3.
        CALL STOR_VAX_INS('BEQL',DLABL,QNUM+1,#OF_GVINS);                      |  4174 CHOOS_BRTYP CASE 4.
      END;  /* OF CASE STATEMENT */                                            |  4218 CHOOS_BRTYP CASE 5.
    ELSE DO;  /* BRANCH IS DONE ON FALSE COMPARISON */                         |  4222 CHOOS_BRTYP
      DO CASE COMP_KIND;                                                       |  4230 CHOOS_BRTYP
        CALL STOR_VAX_INS('BLSS',DLABL,QNUM+1,#OF_GVINS);                      |  4256 CHOOS_BRTYP
        CALL STOR_VAX_INS('BGTR',DLABL,QNUM+1,#OF_GVINS);                      |  4296 CHOOS_BRTYP CASE 0.
        CALL STOR_VAX_INS('BEQL',DLABL,QNUM+1,#OF_GVINS);                      |  4344 CHOOS_BRTYP CASE 1.
        CALL STOR_VAX_INS('BGEQ',DLABL,QNUM+1,#OF_GVINS);                      |  4392 CHOOS_BRTYP CASE 2.
        CALL STOR_VAX_INS('BLEQ',DLABL,QNUM+1,#OF_GVINS);                      |  4440 CHOOS_BRTYP CASE 3.
        CALL STOR_VAX_INS('BNEQ',DLABL,QNUM+1,#OF_GVINS);                      |  4488 CHOOS_BRTYP CASE 4.
      END;  /* OF CASE STATEMENT */                                            |  4536 CHOOS_BRTYP CASE 5.
    END;  /* OF ELSE DO */                                                     |  4544 CHOOS_BRTYP
END CHOOS_BRTYP;                                                               |  4544 CHOOS_BRTYP
/***************************************************************************/|  4550
/*                       PROCEDURE PROC_INDX                               */|  4550
/*                       L3---8--3                                         */|  4550
/*                                                                         */|  4550
/*   THIS PROCEDURE IS TO PROCESS 'INDX' QUAD BY CHECKING TO SEE IF THE    */|  4550
/*SECOND OPERAND OF THE QUAD IS '#' TYPE, IT THEN INSERTS A VAX INSTRUCTION */|  4550
/*'MOVW'. IF IT IS 'T' TYPE, 'IDENT_OPX' IS CALLED AND AN INDEXED REGISTER IS*/|  4550
/*GENERATED, IF IT IS NOT 'T' AND '#' TYPE, THEN 'SYMBTAYL' PROCEDURE IS   */|  4550
/*CALLED AND GENERATES AN INDEXED REGISTER. THE LAST THREE IF STATEMENTS   */|  4550
/*SET A MARK TO INDICATE WHICH OPERAND IN THE NEXT QUAD IS INDEXED. FINALLY */|  4550
/*THE FLAG 'AFT_INDX' IS SET TO 1.                                          */|  4550
/***************************************************************************/|  4550
PROC_INDX:PROCEDURE;                                                           |  4550
  DECLARE (SYMBOL,OP3) CHARACTER;                                              |  4550 PROC_INDX
    IF BYTE(QOP2(QNUM))=BYTE('#') THEN                                         |  4562 PROC_INDX
      DO;                                                                      |  4600 PROC_INDX
      CALL IDENT_OPX(2);                                                       |  4592 PROC_INDX
      INDX_REG,OP3=OPND;                                                       |  4604 PROC_INDX
      VOPRND=SYMBTAYL(QOP2(QNUM))||','||OP3;                                   |  4616 PROC_INDX
      CALL STOR_VAX_INS('MOVW',VOPRND,QNUM,#OF_GVINS);                         |  4672 PROC_INDX
      END;  /* OF IF THEN DO */                                                |  4708 PROC_INDX
    IF BYTE(QOP2(QNUM),1)=BYTE('T') THEN                                       |  4708 PROC_INDX
      DO;                                                                      |  4748 PROC_INDX
      CALL IDENT_OPX(1);                                                       |  4740 PROC_INDX
      INDX_REG=OPND;                                                           |  4752 PROC_INDX
      END;  /* OF IF THEN DO */                                                |  4760 PROC_INDX
    ELSE DO;                                                                   |  4760 PROC_INDX
      SYMBOL=SYMBTAYL(QOP2(QNUM));                                             |  4768 PROC_INDX
      INDX_REG=SRCH_INDX(SYMBOL,INDX#);                                        |  4792 PROC_INDX
                                                                               |  4818 PROC_INDX
```

```
|    IF QOP1(QNUM)=QOP3(QNUM+1) THEN INDXED_OP#=2;  /* 3ST OPERAND IS INDEXED */|  4956 PROC_INDX
|    ST_INDX=1;                                                                 |  5026 PROC_INDX
| END PROC_INDX;                                                                |  5034 PROC_INDX
| /****************************************************************************/|  5040
| /*                        PROCEDURE PROC_MOV                                */|  5040
| /*                        L3---8--1                                         */|  5040
| /*                                                                          */|  5040
| /*   THE PROCEDURE IS TO CONVERT THE QUAD INTO MOVE STATEMENT IN VAX        */|  5040
| /*INSTRUCTION. FIRST EXAMINE TO SEE IF THE 1ST AND 3RD OPERANDS ARE IN THE  */|  5040
| /*SAME DATA TYPE BY IDENTIFYING THE TWO OPERANDS, IF SO, GENERATE A         */|  5040
| /*CORRESPONDING 'MOVE' INSTRUCTION. IF IT IS NOT, INSERT A 'CONVERT'        */|  5040
| /*INSTRUCTION TO CONVERT THE 1ST OPERAND INTO THE DATA TYPE OF 3RD OPERAND, */|  5040
| /*THEN GENERATE THE 'MOVE' INSTRUCTION.                                     */|  5040
| /****************************************************************************/|  5040
| PROC_MOV:PROCEDURE;                                                           |  5040
|   DECLARE (OP1,OP3) CHARACTER;                                                |  5040 PROC_MOV
|     CALL IDENT_OPX(0);            /* IDENTIFY 1ST OPERAND */                  |  5052 PROC_MOV
|     OP1=OPND;                                                                 |  5062 PROC_MOV
|     OP1CT=OPXCT;                                                              |  5070 PROC_MOV
|     CALL IDENT_OPX(2);            /* IDENTIFY 3RD OPERAND */                  |  5078 PROC_MOV
|     OP3=OPND;                                                                 |  5090 PROC_MOV
|     OP3CT=OPXCT;                                                              |  5098 PROC_MOV
|     IF (OP3CT¬='R')&(OP1CT¬='R') THEN                                         |  5106 PROC_MOV
|       IF (OP3CT¬=OP1CT)&(OP1CT¬='#') THEN  /* CONVERT OP1 TO TYPE OF OP3 */   |  5212 PROC_MOV
|          DO;                                                                  |  5330 PROC_MOV
|         GEN_REG#='2';                                                         |  5342 PROC_MOV
|         CALL CONV_DATA(OP1CT||OP3CT,OP1,'2');                                 |  5350 PROC_MOV
|         OP1='R2';                                                             |  5394 PROC_MOV
|         END;  /* OF THEN DO */                                                |  5402 PROC_MOV
|     IF OP3CT='R' THEN                                                         |  5402 PROC_MOV
|       DO;                                                                     |  5452 PROC_MOV
|       IF (OP1CT='R')|(OP1CT='#') THEN OP3CT='W';                              |  5444 PROC_MOV
|         ELSE OP3CT=OP1CT;                                                     |  5570 PROC_MOV
|       END;  /* OF IF THEN ELSE */                                             |  5586 PROC_MOV
|     VOPERAT='MOV'||OP3CT;                                                     |  5586 PROC_MOV
|      VOPRND=OP1||','||OP3;                                                    |  5610 PROC_MOV
|     CALL STOR_VAX_INS(VOPERAT,VOPRND,QNUM,#OF_GVINS);                         |  5650 PROC_MOV
| END PROC_MOV;                                                                 |  5686 PROC_MOV
| /****************************************************************************/|  5692
| /*                        PROCEDURE PROC_UNST                               */|  5692
| /*                        L3---7--8                                         */|  5692
| /*                                                                          */|  5692
| /*   THE PROCEDURE IS TO DETERMINE WHETHER THE QUAD IS THE END OF THE PRO-  */|  5692
| /*GRAM, IF THE 'QIDBUF' OF THE NEXT QUAD IS '75', THEN GENERATE A 'RET' VAX */|  5692
| /*INSTRUCTION. ELSE, GENERATE A 'POPR' VAX INSTRUCTION.                     */|  5692
| /****************************************************************************/|  5692
| PROC_UNST:PROCEDURE;                                                          |  5692
|   DECLARE SYMBOL CHARACTER;                                                   |  5692 PROC_UNST
|     /* CHECK TO SEE IF NEXT QUAD IS RET */                                    |  5704 PROC_UNST
|     IF QIDBUF(QNUM+1)='75' THEN                                               |  5704 PROC_UNST
|       DO;                                                                     |  5766 PROC_UNST
|       SKIP_NEXT_QUAD=1;                                                       |  5758 PROC_UNST
|       CALL STOR_VAX_INS('RET','',QNUM,#OF_GVINS);                             |  5766 PROC_UNST
|       END;  /* NEXT QUAD IS NOT RETN */                                       |  5800 PROC_UNST
|     ELSE DO;  /* NEXT QUAD IS NOT RETN */                                     |  5800 PROC_UNST
|       SYMBOL=SYMBTAYL(QOP3(QNUM));                                            |  5808 PROC_UNST
|       CALL STOR_VAX_INS('POPR',' M<R5>',QNUM,#OF_GVINS);                      |  5832 PROC_UNST
|       END;  /* OF ELSE DO */                                                  |  5868 PROC_UNST
| END PROC_UNST;                                                                |  5868 PROC_UNST
| /****************************************************************************/|  5874
| /*                        PROCEDURE COMP_CHKRR                              */|  5874
| /*                        L3---7--1                                         */|  5874
| /*                                                                          */|  5874
```

```
/*INSTRUCTION. FINALLY EXAMINE THE NEXT QUAD, IF THE 'QIDBUF' IS '07', THEN   */|  5874
/*T    NEXT INSTRUCTION IS A BRANCH ON TRUE COMPARISON. IF T   'QIDBUF' IS     */|  5874
/*'..', THEN THE NEXT INSTRUCTION IS A BRANCH ON FALSE COMPA   SON. IF THE    */|  5874
/*'QIDBUF' IS NEITHER '07' NOR '08', THEN GENERATE A 'MOVE' STATEMENT AND     */|  5874
/*PRINT OUT A ERROR MESSAGE.                                                   */|  5874
/******************************************************************************/|  5874
COMP_CHKBR:PROCEDURE(COMP_TYP);                                                  |  5874
  DECLARE (OP1,OP2) CHARACTER, (NOCONV,COMP_TYP) BIT(8);                        |  5874 COMP_CHKBR
    NOCONV=0;                                                                    |  5886 COMP_CHKBR
    CALL IDENT_OPX(0);  /* IDENTIFY THE 1ST OPERAND */                          |  5892 COMP_CHKBR
    OP1=OPND;                                                                    |  5902 COMP_CHKBR
    OP1CT=OPXCT;                                                                 |  5910 COMP_CHKBR
    IF OP1CT='R' THEN NOCONV=1;                                                  |  5918 COMP_CHKBR
    CALL IDENT_OPX(1);  /* IDENTIFY THE 2ND OPERAND */                          |  5968 COMP_CHKBR
    OP2=OPND;                                                                    |  5980 COMP_CHKBR
    OP2CT=OPXCT;                                                                 |  5988 COMP_CHKBR
    IF OP2CT='R' THEN NOCONV=2;                                                  |  5996 COMP_CHKBR
    IF (OP1CT¬=OP2CT)&(OP1CT¬='#') THEN                                          |  6046 COMP_CHKBR
      IF (OP2CT¬='#')&(NOCONV=0) THEN                                            |  6152 COMP_CHKBR
        DO;  /* CONVERT TYPE OF OP1 TO TYPE OF OP2 */                            |  6242 COMP_CHKBR
        CALL CONV_DATA(OP1CT||OP2CT,OP1,'2');                                   |  6254 COMP_CHKBR
        OP2=OPND||OP1CT;                                                         |  6298 COMP_CHKBR
        END;  /* OF IF THEN DO */                                                |  6322 COMP_CHKBR
    IF (OP2CT='R')|(OP2CT='#') THEN                                              |  6322 COMP_CHKBR
      DO;                                                                        |  6428 COMP_CHKBR
      IF (OP1CT¬='R')&(OP1CT¬='#') THEN OP2CT=OP1CT;                             |  6440 COMP_CHKBR
        ELSE OP2CT='W';                                                          |  6566 COMP_CHKBR
      END;  /* OF IF THEN DO */                                                  |  6582 COMP_CHKBR
    CALL STOR_VAX_INS('CMP'||OP2CT,OP1||','||OP2,QNUM,#OF_GVINS);               |  6582 COMP_CHKBR
    IF QIDBUF(QNUM+1)='07' THEN  /* NEXT INSTRUCTION IS BRT */                   |  6666 COMP_CHKBR
      CALL CHOOS_BRTYP(COMP_TYP,1);                                              |  6728 COMP_CHKBR
    IF QIDBUF(QNUM+1)='08' THEN  /* NEXT INSTRUCTION IS BRF */                   |  6742 COMP_CHKBR
      CALL CHOOS_BRTYP(COMP_TYP,0);                                              |  6804 COMP_CHKBR
    IF (QIDBUF(QNUM+1)¬='07')&(QIDBUF(QNUM+1)¬='08') THEN                        |  6816 COMP_CHKBR
      DO;                                                                        |  6946 COMP_CHKBR
      CALL IDENT_OPX(2);  /* IDENTIFY 3RD OPERAND */                             |  6958 COMP_CHKBR
      CALL STOR_VAX_INS('MOV'||OP2CT,'#1'||','||OPND,QNUM,#OF_GVINS);           |  6970 COMP_CHKBR
      OUTPUT='NEXT QUAD TO THIS COMPARISON MUST BE BRT OR BRF';                  |  7054 COMP_CHKBR
      END;  /* OF IF THEN DO */                                                  |  7074 COMP_CHKBR
END COMP_CHKBR;                                                                  |  7074 COMP_CHKBR
/******************************************************************************/|  7080
/*                        PROCEDURE PROC_ARITH                                */|  7080
/*                        L3---5--1                                           */|  7080
/*                                                                            */|  7080
/*   THIS PROCEDURE PRODUCES THREE MAJOR ARITHMETIC OPERATIONS--ADDITION,     */|  7080
/*SUBSTRACTION AND MULTIPLICATION. 'PROC_THOP_ARITH' IS A SUBPROCEDURE WITHIN*/|  7080
/*THIS PROCEDURE. IDENTIFY THE 1ST AND 3RD OPERANDS FIRST. IF THEY ARE NOT IN*/|  7080
/*THE SAME DATA TYPE, CONVERT THE 1ST OPERAND TO THE TYPE OF 3ND OPERAND,    */|  7080
/*THEN EXAMINE THE 2ND OPERAND. IF IT IS NOT BLANKS, THEN THE QUAD HAS THREE */|  7080
/*OPERANDS. SUBPROCEDURE 'PROC_THOP_AITH' IS CALLED. CALL 'STOR_VAX_INS'     */|  7080
/*PROCEDURE TO STORE THE CORRESPONDING VAX INSTRUCTION.                      */|  7080
/******************************************************************************/|  7080
PROC_ARITH:PROCEDURE(OPERAT);                                                    |  7080
  DECLARE (OP1,OP2,OP3,OPERAT) CHARACTER,NOCONV BIT(8);                         |  7080 PROC_ARITH
  PROC_THOP_ARITH:PROCEDURE(DAT_TYP);                                           |  7092 PROC_ARITH
    DECLARE DAT_TYP CHARACTER;                                                   |  7092 PROC_THOP_ARITH
      CALL IDENT_OPX(1);  /* IDENTIFY THE 2ND OPERAND */                        |  7104 PROC_THOP_ARITH
      OP2=OPND;                                                                  |  7116 PROC_THOP_ARITH
      OP2CT=OPXCT;                                                               |  7124 PROC_THOP_ARITH
      IF (OP2CT¬='R')&(OP2CT¬='#') THEN                                          |  7132 PROC_THOP_ARITH
      IF (OP2CT¬=DAT_TYP)&(DAT_TYP¬='R') THEN                                    |  7238 PROC_THOP_ARITH
        DO;                                                                      |  7356 PROC_THOP_ARITH
        GEN_REG#='3';                                                            |  7368 PROC_THOP_ARITH
        ...(OP2CT||DAT_TYP,OP2,'...');                                           |  7374 PROC_THOP_ARITH
```

```
     IF DAT_TYP='R' THEN                                                   | 7428 PROC_THOP_ARITH
       DO;                                                                 | 7478 PROC_THOP_ARITH
       IF (OP2CT¬='R')|(OP2CT¬='#') THEN OP3CT='W';                        | 7470 PROC_THOP_ARITH
       ELSE OP3CT=OP2CT;                                                   | 7596 PROC_THOP_ARITH
       END;  /* OF IF THEN DO */                                           | 7612 PROC_THOP_ARITH
     VOPRND=OP1||','||OP2||','||OP3;                                       | 7612 PROC_THOP_ARITH
     VOPERAT=OPERAT||OP3CT||'3';                                           | 7684 PROC_THOP_ARITH
     CALL STOR_VAX_INS(VOPERAT,VOPRND,QNUM,#OF_GVINS);                     | 7724 PROC_THOP_ARITH
   END;                                                                    | 7760 PROC_THOP_ARITH
   NOCONV=0;                                                               | 7766 PROC_ARITH
   CALL IDENT_OPX(0);  /* IDENTIFY 1ST OPERAND */                          | 7772 PROC_ARITH
   OP1=OPND;                                                               | 7782 PROC_ARITH
   OP1CT=OPXCT;                                                            | 7790 PROC_ARITH
   IF (OP1CT='#')|(OP1CT='R') THEN NOCONV=1;                               | 7798 PROC_ARITH
   CALL IDENT_OPX(2);  /* IDENTIFY 3RD OPERAND */                          | 7924 PROC_ARITH
   OP3=OPND;                                                               | 7936 PROC_ARITH
   OP3CT=OPXCT;                                                            | 7944 PROC_ARITH
   IF OP3CT='R' THEN NOCONV=2;                                             | 7952 PROC_ARITH
   IF (NOCONV=0)&(OP1CT¬=OP3CT) THEN                                       | 8002 PROC_ARITH
     DO;                                                                   | 8084 PROC_ARITH
     GEN_REG#='2';                                                         | 8096 PROC_ARITH
     CALL CONV_DATA(OP1CT||OP3CT,OP1,'2');                                 | 8104 PROC_ARITH
     OP1='R2';                                                             | 8148 PROC_ARITH
     END;  /* OF IF THEN DO */                                             | 8156 PROC_ARITH
   IF BYTE(QOP2(QNUM),1)¬=BYTE(' ') THEN    /* ADD HAS 3 OPERAND */        | 8156 PROC_ARITH
     CALL PROC_THOP_ARITH(OP3CT);                                          | 8196 PROC_ARITH C20 = 64.
     ELSE DO;                                                              | 8204 PROC_ARITH
         IF OP3CT='R' THEN                                                 | 8212 PROC_ARITH
           DO;                                                             | 8262 PROC_ARITH
           IF (OP1CT='R')|(OP1CT='#') THEN OP3CT='W';                      | 8254 PROC_ARITH
           ELSE OP3CT=OP1CT;                                               | 8380 PROC_ARITH
           END;  /* OF IF THEN DO */                                       | 8396 PROC_ARITH
        VOPRND=OP1||','||OP3;                                              | 8396 PROC_ARITH
        VOPERAT=OPERAT||OP3CT||'2';                                        | 8436 PROC_ARITH
        CALL STOR_VAX_INS(VOPERAT,VOPRND,QNUM,#OF_GVINS);                  | 8476 PROC_ARITH
      END;  /* OF ELSE DO */                                               | 8512 PROC_ARITH
 END PROC_ARITH;                                                           | 8512 PROC_ARITH
 /***********************************************************************/| 8518
 /*                        PROCEDURE LABL_ADDR                         */| 8518
 /*                        L3---4-3                                    */| 8518
 /*                                                                    */| 8518
 /*   THE PROCEDURE IS TO GENERATE A LABLE FOR A VAX INSTRUCTION BY    */| 8518
 /*CONCATENATING THE CHARACTER 'L' WITH THE FIRST OPERAND OF QUAD.     */| 8518
 /***********************************************************************/| 8518
 LABL_ADDR:PROCEDURE;                                                      | 8518
   DECLARE SYMBOL CHARACTER;                                               | 8518 LABL_ADDR
     SYMBOL=SYMBTAYL(QOP1(QNUM));                                          | 8530 LABL_ADDR
     VLABEL_FLD(#OF_GVINS)='L'||SYMBOL||':';                               | 8554 LABL_ADDR
     VQNUM(#OF_GVINS)=QNUM;                                                | 8618 LABL_ADDR
 END LABL_ADDR;                                                            | 8634 LABL_ADDR
 /***********************************************************************/| 8640
 /*                        PROCEDURE SUBENT_ADDR                       */| 8640
 /*                        L3---4--2                                   */| 8640
 /*   THIS PROCEDURE INITIALIZES A VAX ASSEMBLY LANGUAGE PROGRAM. THE PROGRAM */| 8640
 /*LABLE IS GENERATED BY FIRST OPERAND OF THE FIRST QUADRUPLE.         */| 8640
 /***********************************************************************/| 8640
 SUBENT_ADDR:     PROCEDURE;                                               | 8640
   DECLARE SYMBOL CHARACTER;                                               | 8640 SUBENT_ADDR
     VQNUM(#OF_GVINS)=QNUM; /* THIS QUAD WILL CORRESPOND TO TWO LINES OF VAX */ | 8652 SUBENT_ADDR
     SYMBOL=SYMBTAYL(QOP1(QNUM));                                          | 8668 SUBENT_ADDR
     VLABEL_FLD(#OF_GVINS)=SYMBOL||'::';                                   | 8692 SUBENT_ADDR
     #OF_GVINS=#OF_GVINS+1;                                                | 8732 SUBENT_ADDR
     /* ENTRY MASK TO SAVE R2 TO R5 */                                     | 8744 SUBENT_ADDR
```

```
/*****************************************************************************.  ....
/*                       PROCEDURE ALLOCATION                                */|  8786
/*                       L3---4--1                                           */|  8786
/*                                                                           */|  8786
/*   THIS PROCEDURE IS TO DEFINE A STORAGE OR A CONSTANT IN VAX ASSEMBLY     */|  8786
/*LANGUAGE. THE LABLE IS GENERATED BY THE FIRST OPERAND OF QUAD. THE OPERATOR*/|  8786
/*IS GENERATED BY EITHER 'BTYPE' OR 'STYPE' WHICH IS A PARAMETER PASSED BY   */|  8786
/*THE CALLING PROCEDURE. THE OPERAND FILED IS GENERATED BY EITHER THE 2ND OR */|  8786
/*GENERATED BY THE 2ND OPERAND IN NEXT QUAD. IF IT IS PRODUCED BY 'STYPE', IT*/|  8786
/*3RD OPERAND OF THE NEXT QUAD. IF OPERATOR IS PRODUCED BY 'BTYPE', IT IS    */|  8786
/*IS GENERATED BY THE 3RD OPERAND IN NEST QUAD. AFTER THE CONVERSION. THE    */|  8786
/*LABLE AND DATA TYPE ARE STORED FOR FUTURE USE.                             */|  8786
/*****************************************************************************/|  8786
ALLOCATION:PROCEDURE(BTYPE,STYPE);                                             |  8786
  DECLARE (BTYPE,STYPE,SYMBOL,SUBS) CHARACTER;                                 |  8786 ALLOCATION
    SKIP_NEXT_QUAD=1;     /* TWO QUADS ARE PROCESSED */                        |  8798 ALLOCATION
    SYMBOL=SYMBTAYL(QOP1(QNUM));                                               |  8806 ALLOCATION
    VLABEL_FLD(DECL#)=SYMBOL||':';                                             |  8830 ALLOCATION
    IF BYTE(QOP3(QNUM+1),1)=BYTE(' ') THEN  /* SYMBOL DOES NOT HAVE INIT VALU*/|  8870 ALLOCATION
      DO;                                                                      |  8914 ALLOCATION
        VOPERAT_FLD(DECL#)=BTYPE;                                              |  8906 ALLOCATION
        VOPRND_FLD(DECL#)=SYMBTAYL(QOP2(QNUM+1));                              |  8922 ALLOCATION
      END;  /* OF IF THEN DO */                                                |  8966 ALLOCATION
    ELSE DO;  /* VARIABLE HAS INITIAL VALUE */                                 |  8966 ALLOCATION
      VOPERAT_FLD(DECL#)=STYPE;                                                |  8974 ALLOCATION
      SUBS=SUBSTR(QOP3(QNUM+1),1);  /* DISCARD # */                            |  8990 ALLOCATION
      VOPRND_FLD(DECL#)=SYMBTAYL(SUBS);                                        |  9024 ALLOCATION C21 = 16777215.
      END;  /* OF ELSE DO */                                                   |  9056 ALLOCATION
      /* STORE SYMBOL NAME AND ITS DATA TYPE FOR FUTURE USE */                 |  9056 ALLOCATION
    NAME(#OF_DEFSYMB)=SYMBOL;                                                  |  9056 ALLOCATION
    CTYPE(#OF_DEFSYMB)=SUBSTR(STYPE,1,1);                                      |  9072 ALLOCATION
    #OF_DEFSYMB=#OF_DEFSYMB+1;                                                 |  9096 ALLOCATION
    DECL#=DECL#-1;                                                             |  9108 ALLOCATION
END ALLOCATION;                                                                |  9120 ALLOCATION
/*****************************************************************************/|  9126
/*                       PROCEDURE PROC_BG                                   */|  9126
/*                       L3---2--5                                           */|  9126
/*                                                                           */|  9126
/*   THIS PROCEDURE PROCESSES THE BRANCH ON GREATER THEN. FIRST IDENTIFY THE */|  9126
/*2ND AND 3RD OPERANDS. IF THEY ARE NOT THE SAME DATA TYPE, CONVERT 2ND      */|  9126
/*OPERAND TO THE TYPE OF 3RD OPERAND BY INSERTING A 'CONVERT' INSTRUCTION,   */|  9126
/*THEN GENERATE A 'COMPARE' INSTRUCTION. FINALLY PROCEDURE 'GLABL' IS CALLED */|  9126
/*TO ASSIGN A LABLE TO THE BRANCH ADDRESS. A 'BGTR' INSTRUCTION IS GENERATED.*/|  9126
/*****************************************************************************/|  9126
PROC_BG:    PROCEDURE;                                                         |  9126
  DECLARE (OP2,OP3,OLABL) CHARACTER, QUAD# FIXED, NOCONV BIT(8) INITIAL(0);    |  9126 PROC_BG
    CALL IDENT_OPX(1);        /* IDENTIFY 2ND OPERAND */                       |  9138 PROC_BG
    OP2=OPND;                                                                  |  9150 PROC_BG
    OP2CT=OPXCT;                                                               |  9158 PROC_BG
    IF OP2CT='R' THEN NOCONV=2;                                                |  9166 PROC_BG
    CALL IDENT_OPX(2);        /* IDENTIFY 3RD OPERAND */                       |  9216 PROC_BG
    OP3=OPND;                                                                  |  9228 PROC_BG
    OP3CT=OPXCT;                                                               |  9236 PROC_BG
    IF OP3CT='R' THEN NOCONV=3;                                                |  9244 PROC_BG
    IF (NOCONV=0)&(OP3CT¬='#') THEN                                            |  9294 PROC_BG
      IF (OP2CT¬=OP3CT)&(OP2CT¬='#') THEN  /* CONVERT OP1 TO TYPE OF OP3 */    |  9376 PROC_BG
        DO;                                                                    |  9494 PROC_BG
        GEN_REG#='2';                                                          |  9506 PROC_BG
        CALL CONV_DATA(OP2CT||OP3CT,OP2,'2');                                  |  9514 PROC_BG
        OP2='R2';                                                              |  9558 PROC_BG
        END;  /* OF IF THEN DO */                                              |  9566 PROC_BG
    IF (OP3CT='R')|(OP3CT='#') THEN                                            |  9566 PROC_BG
      DO;                                                                      |  9672 PROC_BG
        IF (OP2CT¬='R')&(OP2CT¬='#') THEN OP3CT=OP2CT;                         |  9684 PROC_BG
                                                                               |  9810 PROC_BG
```

```
|    V)PRN)=OP2||','||OP3;                                                      |  9850 PROC_BG
|    _ALL STOR_VAX_INS(VOPERAT,VOPRND, QNUM,#OF_GVINS);                         |  9890 PROC_BG
|    _ABL=GLABL(QOP1(QNUM));                                                    |  9926 PROC_BG
|    CALL STOR_VAX_INS('BGTR',DLABL,QNUM,#OF_GVINS);                            |  9950 PROC_BG
| END PROC_BG;                                                                  |  9986 PROC_BG
| /***************************************************************************/|  9992
| /*                          PROCEDURE PROC_BR                              */|  9992
| /*                          L3---2--1                                      */|  9992
| /*                                                                         */|  9992
| /*   PROC_BR PROCEDURE PASSES THE 1ST OPERAND OF QUAD TO PROCEDURE 'GLABL'  */|  9992
| /*TO PRODUCE A LABLE FOR VAX BRANCH INSTRUCTION.                           */|  9992
| /***************************************************************************/|  9992
| PROC_BR:PROCEDURE;                                                            |  9992
|   DECLARE DLABL CHARACTER;                                                    |  9992 PROC_BR
|     DLABL=GLABL(QOP1(QNUM));                                                  | 10004 PROC_BR
|       /* FILL THE OUTPUT FILE */                                              | 10028 PROC_BR
|     CALL STOR_VAX_INS('BRW',DLABL,QNUM,#OF_GVINS);                            | 10028 PROC_BR
| END PROC_BR;                                                                  | 10064 PROC_BR
| /***************************************************************************/| 10070
| /*                          PROCEDURE SRCH_FBRDESQ                         */| 10070
| /*                          L3---1--1                                      */| 10070
| /*                                                                         */| 10070
| /*   INPUT PARAMETER 'FQNUM' IS EQUAL TO 'QNUM' WHICH IS EQUAL TO THE NUMBER */| 10070
| /*OF QUADS BEING PROCESSED BY PROCEDURE 'DRIVER'. IF THE 'FQNUM' IS ONE     */| 10070
| /*ELEMENT IN THE ARRAY 'FBRDESQ(I)', THEN SET 'FOUND' EQUAL TO 1 AND RETURN  */| 10070
| /*IT TO CALLING PROCEDURE, OTHERWISE RETURN -1.                            */| 10070
| /***************************************************************************/| 10070
| SRCH_FBRDESQ:PROCEDURE(FQNUM) BIT(8);                                         | 10070
|   /* SEARCHES LABEL NUMBER OF THE REFERENCED QUAD */                          | 10070 SRCH_FBRDESQ
|   DECLARE I BIT(8),FQNUM FIXED, FOUND BIT(1);                                 | 10070 SRCH_FBRDESQ
|     I=0;                                                                      | 10082 SRCH_FBRDESQ
|     FOUND=0;                                                                  | 10088 SRCH_FBRDESQ
|     DO WHILE (FOUND=0)&(I<=FBRDESQ_IND);                                      | 10094 SRCH_FBRDESQ
|       IF FBRDESQ(I)=FQNUM THEN FOUND=1;                                       | 10164 SRCH_FBRDESQ
|       I=I+1;                                                                  | 10198 SRCH_FBRDESQ
|     END;  /* OF DO WHILE */                                                   | 10212 SRCH_FBRDESQ
|     IF FOUND THEN RETURN I-1;                                                 | 10220 SRCH_FBRDESQ
|     ELSE RETURN -1;                                                           | 10250 SRCH_FBRDESQ
| END SRCH_FBRDESQ;                                                             | 10268 SRCH_FBRDESQ
| /***************************************************************************/| 10274
| /*                          PROCEDURE MISC                                 */| 10274
| /*                          L2---3--9                                      */| 10274
| /*                                                                         */| 10274
| /*   THIS PROCEDURE PROCESSES MISCELLANEOUS OPERATIONS.                    */| 10274
| /***************************************************************************/| 10274
| MISC:PROCEDURE(QINGRP);                                                       | 10274
|   DECLARE QINGRP FIXED ;                                                      | 10274 MISC
|     DO CASE QINGRP;                                                           | 10286 MISC
|     /* CALL PROC_NAME;     /* CASE 0 */                                       | 10310 MISC
|       OUTPUT=0;                                                               | 10310 MISC
|     /*                                                                        | 10338 MISC CASE 0.
|       CALL PROC_PUSH;     /* CASE 1 */                                        | 10338 MISC
|       OUTPUT=1;                                                               | 10338 MISC
|     /*                                                                        | 10376 MISC CASE 1.
|       CALL PROC_POP;     0 * CASE 2 */                                        | 10376 MISC
|       OUTPUT=2;                                                               | 10376 MISC
|     /*                                                                        | 10414 MISC CASE 2.
|       CALL PROC_HALF;     /* CASE 3 */                                        | 10414 MISC
|       OUTPUT=3;                                                               | 10414 MISC
|     /*                                                                        | 10452 MISC CASE 3.
|       CALL PROC_LINK;     /* CASE 4 */                                        | 10452 MISC
|       OUTPUT=4;                                                               | 10452 MISC
|       CALL STOR_VAX_INS('RET','',QNUM,#OF_GVINS);                             | 10490 MISC CASE 4.
|     /*                                                                        | 10532 MISC CASE 5.
```

```
      OUTPUT=0;
      CALL PROC_UNST;     /* CASE 7 */
   END;  /* OF CASE STATEMENT */                                                | 10570 MISC CASE 6.
END MISC;                                                                       | 10586 MISC CASE 7.
/*****************************************************************************/| 10594 MISC
/*                     PROCEDURE MOVE_CONV                                   */| 10600
/*                     L2---3--8                                             */| 10600
/*                                                                           */| 10600
/*   THIS PROCEDURE IS TO CHECK TO SEE IF THE QIDBUF=60, THEN CALL PROC_MOV  */| 10600
/*TO PROCESS THE CONVERSION OF MOVE STATEMENT. IF IT IS EQUAL TO 61, PRINT   */| 10600
/*OUT 'QIDBUF IS 61, NOT IMPLEMENTED YET'. IF IT IS 62, CALL PROC_INDX       */| 10600
/*PROCEDURE.                                                                 */| 10600
/*****************************************************************************/| 10600
MOVE_CONV:PROCEDURE(QINGRP);                                                    | 10600
  DECLARE QINGRP BIT(8);                                                        | 10600 MOVE_CONV
    DO CASE QINGRP;                                                             | 10612 MOVE_CONV
      CALL PROC_MOV;                                                            | 10638 MOVE_CONV
      OUTPUT='QIDBUF IS 61, NOT IMPLEMENTED YET';                               | 10646 MOVE_CONV CASE 0.
      CALL PROC_INDX;                                                           | 10674 MOVE_CONV CASE 1.
    END;  /* OF CASE STATEMENT */                                               | 10690 MOVE_CONV CASE 2.
END MOVE_CONV;                                                                  | 10698 MOVE_CONV
/*****************************************************************************/| 10704
/*                     PROCEDURE LOGIC_OP                                    */| 10704
/*                     L2---3--7                                             */| 10704
/*****************************************************************************/| 10704
LOGIC_OP:PROCEDURE(QINGRP);                                                     | 10704
  DECLARE QINGRP BIT(8);                                                        | 10704 LOGIC_OP
    DO CASE QINGRP;                                                             | 10716 LOGIC_OP
      /*                                                                        | 10742 LOGIC_OP
      CALL PROC_AND;                    /* CASE 0 */                            | 10742 LOGIC_OP
      OUTPUT=13;                                                                | 10742 LOGIC_OP
      /*                                                                        | 10772 LOGIC_OP CASE 0.
      CALL PROC_OR;                     /* CASE 1 */                            | 10772 LOGIC_OP
      OUTPUT=12;                                                                | 10772 LOGIC_OP
      /*                                                                        | 10810 LOGIC_OP CASE 1.
      CALL PROC_CMPL;                   /* CASE 2 */                            | 10810 LOGIC_OP
      OUTPUT=11;                                                                | 10810 LOGIC_OP
      /*                                                                        | 10848 LOGIC_OP CASE 2.
      CALL PROC_XOR;                    /* CASE 3 */                            | 10848 LOGIC_OP
      OUTPUT=10;                                                                | 10848 LOGIC_OP
      CALL COMP_CHKBR(QINGRP-4);        /* CASE 4 */                            | 10886 LOGIC_OP CASE 3.
      CALL COMP_CHKBR(QINGRP-4);        /* CASE 5 */                            | 10916 LOGIC_OP CASE 4. C22 = 4.
      CALL COMP_CHKBR(QINGRP-4);        /* CASE 6 */                            | 10946 LOGIC_OP CASE 5.
      CALL COMP_CHKBR(QINGRP-4);        /* CASE 7 */                            | 10976 LOGIC_OP CASE 6.
      CALL COMP_CHKBR(QINGRP-4);        /* CASE 8 */                            | 11006 LOGIC_OP CASE 7.
      CALL COMP_CHKBR(QINGRP-4);        /* CASE 9 */                            | 11036 LOGIC_OP CASE 8.
    END;  /* OF CASE STATEMENT */                                               | 11066 LOGIC_OP CASE 9.
END LOGIC_OP;                                                                   | 11074 LOGIC_OP
/*****************************************************************************/| 11080
/*                     PROCEDURE SHIFT_OP                                    */| 11080
/*                     L2---3--6                                             */| 11080
/*                                                                           */| 11080
/*   THIS PROCEDURE IS TO CONVERT SHIFT INSTRUCTION OF QUAD INTO             */| 11080
/*MULTIPLICATION OF VAX INSTRUCTION. THE OPERATOR IS DECIDED BY THE TYPE OF  */| 11080
/*SECOND OPERAND OF QUAD. IF IT IS B-TYPE, THEN OPERATOR IS 'MULB3'. IF IF IS*/| 11080
/*W-TYPE, THEN THE OPERATOR IS 'MULW3'. IF IT IS R, #, OR L-TYPE, THEN       */| 11080
/*OPERATOR IS 'MULL3'.                                                       */| 11080
/*****************************************************************************/| 11080
SHIFT_OP:PROCEDURE(QINGRP);                                                     | 11080
  DECLARE (QINGRP,SHIFT_SIZ,I) BIT(8),MULTIPLIER FIXED,                         | 11080 SHIFT_OP
          (OP1,OP2,OP3) CHARACTER;                                              | 11092 SHIFT_OP
    MULTIPLIER=1;                                                               | 11092 SHIFT_OP
    CALL IDENT_OPX(1);  /* IDENTIFY 2ND OPERAND */                              | 11100 SHIFT_OP
    OP2=OPND;                                                                   | 11112 SHIFT_OP
                                                                                | 11120 SHIFT_OP
```

```
      SHIFT_SIZ=[illegible]
      DO I=1 TO SHIFT_SIZ;                                                     | 11182 SHIFT_OP
        MULTIPLIER=MULTIPLIER*2;                                               | 11230 SHIFT_OP
      END;  /* OF DO LOOP */                                                   | 11242 SHIFT_OP
      OP1='#'||MULTIPLIER;                                                     | 11250 SHIFT_OP
      CALL IDENT_OPX(2);  /* IDENTIFY 3RD OPERAND */                           | 11282 SHIFT_OP
      OP3=OPND;                                                                | 11294 SHIFT_OP
      IF (OP2CT='R')|(OP2CT='#')|(OP2CT='L') THEN                              | 11302 SHIFT_OP
      VOPERAT='MULL3';                                                         | 11464 SHIFT_OP
      IF OP2CT='B' THEN VOPERAT='MULB3';                                       | 11484 SHIFT_OP
      IF OP2CT='W' THEN VOPERAT='MULW3';                                       | 11534 SHIFT_OP
      CALL STOR_VAX_INS(VOPERAT,OP1||','||OP2||','||OP3,QNUM,#OF_GVINS);       | 11584 SHIFT_OP
  END SHIFT_OP;                                                                | 11684 SHIFT_OP
  /****************************************************************************/| 11690
  /*                      PROCEDURE ARITH_OP                                  */| 11690
  /*                      L2---3--5                                           */| 11690
  /*                                                                          */| 11690
  /*   THIS PROCEDURE CALLS PROCEDURE 'PROC_ARITH' TO PROCESS THE OPERATIONS OF*/| 11690
  /*ADDITION, SUBSTRACTION AND MULTIPLICATION RESPECTIVELY DEPENDING ON THE   */| 11690
  /*VALUE OF 'QINGRP' PASSED BY THE CALLING PROCEDURE.                        */| 11690
  /****************************************************************************/| 11690
  ARITH_OP:PROCEDURE(QINGRP);                                                  | 11690
    DECLARE QINGRP BIT(8);                                                     | 11690 ARITH_OP
      DO CASE QINGRP;                                                          | 11702 ARITH_OP
      CALL PROC_ARITH('ADD');                                                  | 11728 ARITH_OP
      CALL PROC_ARITH('SUB');                                                  | 11744 ARITH_OP CASE 0.
      CALL PROC_ARITH('MUL');                                                  | 11768 ARITH_OP CASE 1.
                     /*                                                        | 11792 ARITH_OP CASE 2.
      CALL PROC_DIV;                                                           | 11792 ARITH_OP
      CALL PROC_MOD;                                                           | 11792 ARITH_OP
                     */                                                        | 11792 ARITH_OP
    END;  /* OF CASE STATEMENT */                                              | 11792 ARITH_OP
  END ARITH_OP;                                                                | 11800 ARITH_OP
  /****************************************************************************/| 11806
  /*                      PROCEDURE DECLARATION                               */| 11806
  /*                      L2---3--4                                           */| 11806
  /*                                                                          */| 11806
  /*   THIS PROCEDURE IS TO CHECK TO SEE IF THE QUAD DEFINES A STORAGE OR A   */| 11806
  /*LABLE BY EXAMINING THE NEXT QUAD. IF THE FIRST DIGIT OF 'QIDBUF'IS '2',   */| 11806
  /*THEN THE QUAD DEFINES A STORAGE. IF IT IS NOT '2', THEN THE QUAD DEFINES  */| 11806
  /*A LABLE. IF THE 'QIDBUF' OF NEXT QUAD IS EQUAL TO 24, THE IT DEFINES A NEW */| 11806
  /*SECTION.                                                                  */| 11806
  /****************************************************************************/| 11806
  DECLARATION:PROCEDURE(QINGRP);                                               | 11806
    DECLARE(QINGRP,NQGRP,NQINGRP) BIT(8);                                      | 11806 DECLARATION
      NVAL_OF_QID=CONVERT_TO_INTEG(QIDBUF(QNUM+1));  /* IDENTIFY NEXT QUAD */  | 11818 DECLARATION
      NQGRP=NVAL_OF_QID/10;                                                    | 11846 DECLARATION
      IF NQGRP=2 THEN                                                          | 11864 DECLARATION
        DO;                                                                    | 11890 DECLARATION
        NQINGRP=NVAL_OF_QID MOD 10;                                            | 11882 DECLARATION
        DO CASE NQINGRP;                                                       | 11902 DECLARATION
          CALL ALLOCATION('.BLKB','.BYTE');  /* BYTE */                        | 11928 DECLARATION
          CALL ALLOCATION('.BLKB','.BYTE');  /* BOOL */                        | 11952 DECLARATION CASE 0.
          CALL ALLOCATION('.BLKL','.LONG');  /* FULL */                        | 11984 DECLARATION CASE 1.
          CALL ALLOCATION('.BLKW','.WORD');  /*HALF */                         | 12016 DECLARATION CASE 2.
          CALL SUBENT_ADDR;    /* NEW PSECT */                                 | 12048 DECLARATION CASE 3.
        END;  /* OF CASE STATEMENT */                                          | 12064 DECLARATION CASE 4.
      END;  /* OF IF THEN DO */                                                | 12072 DECLARATION
      /* THIS ADDR DEFINES A LABEL */                                          | 12072 DECLARATION
      ELSE CALL LABL_ADDR;                                                     | 12072 DECLARATION
  END DECLARATION;                                                             | 12088 DECLARATION
  /****************************************************************************/| 12094
  /*                      PROCEDURE DIR_BRCH                                  */| 12094
  /*                      L2---3--2                                           */| 12094
```

```
   DECL...                                                                    |
     DO CASE QINGRP;                                                          | 12106 DIR_BRNCH
       CALL PROC_BR;                                                          | 12132 DIR_BRNCH
                  /*                                                          | 12140 DIR_BRNCH CASE 0.
       CALL PROC_BZ;                                                          | 12140 DIR_BRNCH
       CALL PROC_BP;                                                          | 12140 DIR_BRNCH
       CALL PROC_BM;                                                          | 12140 DIR_BRNCH
                  */                                                          | 12140 DIR_BRNCH
       OUTPUT='NO PROC_BZ YET';                                               | 12140 DIR_BRNCH
       OUTPUT='NO PROC_BP YET';                                               | 12168 DIR_BRNCH CASE 1.
       OUTPUT='NO PROC_BM YET';                                               | 12196 DIR_BRNCH CASE 2.
       CALL PROC_BG;                                                          | 12224 DIR_BRNCH CASE 3.
                  /*                                                          | 12240 DIR_BRNCH CASE 4.
       CALL PROC_BL;                                                          | 12240 DIR_BRNCH
       CALL PROC_BE;                                                          | 12240 DIR_BRNCH
       CALL PROC_BRT;                                                         | 12240 DIR_BRNCH
       CALL PROC_BRF;                                                         | 12240 DIR_BRNCH
                  */                                                          | 12240 DIR_BRNCH
     END;  /* OF CASE STATEMENT */                                            | 12240 DIR_BRNCH
 END DIR_BRNCH;                                                               | 12248 DIR_BRNCH
 /*************************************************************************/| 12254
 /*                    PROCEDURE FBRLABL_GEN                               */| 12254
 /*                    L2---3--1                                           */| 12254
 /*                                                                        */| 12254
 /*   THIS PROCEDURE CALLS PROCEDURE 'SRCH_FBRDESQ' AND ASSIGNS A VALUE TO */| 12254
 /*'FLABL#'. IF THIS DOESN'T EQUAL TO '-1', THEN A LABLE IS GENERATED.     */| 12254
 /*'VLABEL_FLD' IS SET TO EQUAL TO 'LBLF' CONCATENATED WITH A ONE BYTE VALUE */| 12254
 /*OF FILE DIGIT BASED ON THE VALUE OF 'FLABL#' AND FOLLOWED BY A COLON TO  */| 12254
 /*INDICATE A LABLE. IF 'FLAB#=-1, THEN ERROR MESSAGE IS GENERATED.        */| 12254
 /*************************************************************************/| 12254
 FBRLABL_GEN:PROCEDURE(FQUAD);                                                | 12254
   DECLARE FQUAD FIXED;                                                       | 12254 FBRLABL_GEN
     /* IF QUAD HAS BEEN REFERENCED BEFORE, LABEL IS GENERATED FOR */         | 12266 FBRLABL_GEN
     FLABL#=SRCH_FBRDESQ(FQUAD);                                              | 12266 FBRLABL_GEN
     IF FLABL#¬=-1 THEN                                                       | 12286 FBRLABL_GEN
       DO;  /* GENERATE LABEL */                                              | 12312 FBRLABL_GEN
       IF SKIP_NEXT_QUAD=1 THEN VLABEL_FLD(#OF_GVINS-1)='LBLF'||SUBSTR(DIGIT, | 12304 FBRLABL_GEN
                                                        FLABL#,1)||':';       | 12330 FBRLABL_GEN
         ELSE VLABEL_FLD(#OF_GVINS)='LBLF'||SUBSTR(DIGIT,FLABL#,1)||':';      | 12420 FBRLABL_GEN
       END;  /* OF IF THEN DO */                                              | 12522 FBRLABL_GEN
     ELSE OUTPUT='FORWARD BRANCH DESTINATION QUAD NOT FOUND';                 | 12522 FBRLABL_GEN
 END FBRLABL_GEN;                                                             | 12550 FBRLABL_GEN
 /*************************************************************************/| 12556
 /*                    PROCEDURE OUTPUT_VASM                               */| 12556
 /*                    L1---M--4                                           */| 12556
 /*                                                                        */| 12556
 /*   THIS PROCEDURE IS TO PRINT OUT THE VAX ASSEMBLY LANGUAGE AFTER BEING */| 12556
 /*CONVERTED FROM QUADS. PROCEDURE 'BLANK' IS CALLED TO ALLIGN THE FORMAT OF */| 12556
 /*OUTPUT STYLE.                                                           */| 12556
 /*************************************************************************/| 12556
 OUTPUT_VASM:PROCEDURE(INIT,FINAL,STAT#);                                     | 12556
   DECLARE (VBUFFER,S1,S2,S3,INS#) CHARACTER, (N,INIT,FINAL,STAT#) FIXED;     | 12556 OUTPUT_VASM
     DO N=INIT TO FINAL;                                                      | 12568 OUTPUT_VASM
       VBUFFER='    ';                                                        | 12612 OUTPUT_VASM
       S1=BLANK(VLABEL_FLD(N),8);    /* FORMAT LABEL */                       | 12620 OUTPUT_VASM
       S2=BLANK(VOPERAT_FLD(N),8);   /* FORMAT OPERATION FIELD */             | 12652 OUTPUT_VASM
       S3=BLANK(VOPRND_FLD(N),13);   /* FORMAT OPERAND FIELD */               | 12684 OUTPUT_VASM
       VBUFFER=VBUFFER||S1||S2||S3;                                           | 12716 OUTPUT_VASM
       STAT#=STAT#+1;                                                         | 12772 OUTPUT_VASM
       INS#=STAT#||'      ';                                                  | 12784 OUTPUT_VASM
       INS#=SUBSTR(INS#,0,5);                                                 | 12816 OUTPUT_VASM
       OUTPUT=INS#||VBUFFER;                                                  | 12832 OUTPUT_VASM
     END;  /* OF DO N=INIT TO FINAL */                                        | 12870 OUTPUT_VASM
 END OUTPUT_VASM;                                                             | 12878 OUTPUT_VASM
 ...**********************************************************************/| 1288.
```

```
/*                      L1---1---)
/*                                                                         */| 12884
/*   'SQUADS# IS THE NUMBER OF QUADS TO BE PROCESSED.                      */| 12884
/*    THIS PROCEDURE IS TO INDIVIDUALLY PROCESS THE QUADS. T..RE ARE EIGHT */| 12884
/*PROCEDURES TO BE CALLED TO CONVERT EACH QUAD INTO VAX INSTRUCTION BASED ON */| 12884
/*DIFFERENT TYPE OF QUADS SHOWN BY THE INDEX NMUNBER--'QIDBUF'.            */| 12884
/*************************************************************************/| 12884
DRIVER:PROCEDURE(SQUADS#);                                                   | 12884
  DECLARE SQUADS# FIXED;                                                     | 12884 DRIVER
  /* PROCESS QUADS STORED IN QUAD BUFFERS    */                              | 12896 DRIVER
    DO QNUM=0 TO SQUADS#-1;                                                  | 12896 DRIVER
      IF QBRFLG(QNUM)=1 THEN CALL FBRLABL_GEN(QNUM);                         | 12942 DRIVER
      IF SKIP_NEXT_QUAD=0 THEN                                               | 12980 DRIVER
        DO; /* IF SKIP_NEXT_QUAD IS TRUE THEN QUAD HAS ALREADY BEEN PROCESSED*/| 13006 DRIVER
        NVAL_OF_QID=CONVERT_TO_INTEG(QIDBUF(QNUM));                          | 12998 DRIVER
        QUAD#_IN_GROUP=NVAL_OF_QID MOD 10;                                   | 13022 DRIVER
        QUAD_GROUP=NVAL_OF_QID/10;                                           | 13042 DRIVER
        DO CASE QUAD_GROUP; /* DECIDE ACCORDING TO THE GROUP OF QUAD */      | 13060 DRIVER
          CALL DIR_BRNCH(QUAD#_IN_GROUP);  /* CASE 0 */                      | 13086 DRIVER
          OUTPUT='NO INDIRECT BRANCH HAS BEEN IMPLEMENTED YET'; /* CASE 1 */ | 13104 DRIVER CASE 0.
          CALL DECLARATION(QUAD#_IN_GROUP);  /* CASE 2 */                    | 13132 DRIVER CASE 1.
          CALL ARITH_OP(QUAD#_IN_GROUP);  /* CASE 3 */                       | 13158 DRIVER CASE 2.
          CALL SHIFT_OP(QUAD#_IN_GROUP);  /* CASE 4 */                       | 13184 DRIVER CASE 3.
          CALL LOGIC_OP(QUAD#_IN_GROUP);  /* CASE 5 */                       | 13210 DRIVER CASE 4.
          CALL MOVE_CONV(QUAD#_IN_GROUP);  /* CASE 6 */                      | 13236 DRIVER CASE 5.
          CALL MISC(QUAD#_IN_GROUP);     /* CASE 7 */                        | 13262 DRIVER CASE 6.
        END;  /* OF CASE STATEMENT */                                        | 13288 DRIVER CASE 7.
      END;  /* OF IF THEN DO */                                              | 13296 DRIVER
      ELSE SKIP_NEXT_QUAD=0;                                                 | 13296 DRIVER
    END;  /* OF DO QNUM=0 */                                                 | 13310 DRIVER
END DRIVER;                                                                  | 13318 DRIVER
/*************************************************************************/| 13324
/*                       PROCEDURE FIND_INDX                               */| 13324
/*                        L1---M--2                                        */| 13324
/*                                                                         */| 13324
/*   'QUAD_NUM' IS THE NUMBER OF QUADS TO BE READ IN.                      */| 13324
/*   THE PROCEDURE IS TO EXAMINE EACH ENTRY IN 'QIDBUF' TO FIND IF ITS VALUE */| 13324
/*IS ='62', THEN EXAMINE LEADING BYTE IN 2ND OPERAND(QOP2) AND IF IT IS ='U',*/| 13324
/*THEN CALL PROCEDURE 'SYMBTAYL' TO GENERATE A VARIABLE SYMBOL, STORE INTO AN*/| 13324
/*ARRAY 'INDX_SYMB(INDX#), INCREMENT THE INDX# BY ONE.                     */| 13324
/*************************************************************************/| 13324
FIND_INDX:PROCEDURE(QUAD_NUM);                                               | 13324
  DECLARE (QUAD_NUM,I) FIXED, SYMBOL CHARACTER;                              | 13324 FIND_INDX
    DO I = 0 TO QUAD_NUM;                                                    | 13336 FIND_INDX
      IF QIDBUF(I)='62' THEN                                                 | 13378 FIND_INDX
        DO;                                                                  | 13436 FIND_INDX
      IF BYTE(QOP2(I),1)=BYTE('U') THEN                                      | 13428 FIND_INDX
          DO;                                                                | 13468 FIND_INDX
          SYMBOL=SYMBTAYL(QOP2(I));                                          | 13460 FIND_INDX
          INDX_SYMB(INDX#)=SYMBOL;                                           | 13484 FIND_INDX
          INDX#=INDX#+1;                                                     | 13502 FIND_INDX
          END;  /* OF IF THEN DO */                                          | 13516 FIND_INDX
        END;  /* OF IF THEN DO */                                            | 13516 FIND_INDX
    END;  /* OF DO I=0  */                                                   | 13516 FIND_INDX
END FIND_INDX;                                                               | 13524 FIND_INDX
/*************************************************************************/| 13530
/*                       PROCEDURE INPUT_QUADS                             */| 13530
/*                       L1---M--1                                         */| 13530
/*                                                                         */| 13530
/*   THE PROCEDURE IS TO READ IN AND ECHO-PRINT THE INPUT QUADS AS WELL STORE*/| 13530
/*THEM IN THE FOLLOWING FORMAT:                                            */| 13530
/*           QUADNAME=COLUMN 1-4                                           */| 13530
/*           QUADTYPE=COLUMN  5                                            */| 13530
/*           QOP1     =COLUMN 7-17                                         */| 13530
```

```
| /*
| /***************************************************************************/| 13530
| INF    QUADS:PROCEDURE;                                                    | 13530
|   DECLARE QRECORD CHARACTER;                                               | 13530 INPUT_QUADS
|     /* READ, STORE AND ECHO PRINT QUADS, MAXIMMUM 100 QUADS ARE STORED */  | 13542 INPUT_QUADS
|     OUTPUT='INPUT QUADS:';                                                 | 13542 INPUT_QUADS
|     OUTPUT,QRECORD=INPUT;  /* GET A QUAD */                                | 13562 INPUT_QUADS
|     DO WHILE(LENGTH(QRECORD)>0) & (#OF_QUADS<=QBUFSIZE);                   | 13644 INPUT_QUADS
|       QUADNAME(#OF_QUADS)=SUBSTR(QRECORD,1,4);                             | 13724 INPUT_QUADS C23 = 50.
|       QUADTYPE(#OF_QUADS)=SUBSTR(QRECORD,5,1);                             | 13748 INPUT_QUADS C24 = 50331648.
|       QOP1(#OF_QUADS)=SUBSTR(QRECORD,7,11);                                | 13772 INPUT_QUADS
|       QOP2(#OF_QUADS)=SUBSTR(QRECORD,19,11);                               | 13796 INPUT_QUADS C25 = 167772160.
|       QOP3(#OF_QUADS)=SUBSTR(QRECORD,31,11);                               | 13820 INPUT_QUADS
|       QIDBUF(#OF_QUADS)=SUBSTR(QRECORD,43,2);                              | 13844 INPUT_QUADS
|       #OF_QUADS=#OF_QUADS+1;                                               | 13868 INPUT_QUADS C26 = 16777216.
|       OUTPUT,QRECORD=INPUT;  /* GET A NEW QUAD */                          | 13880 INPUT_QUADS
|     END;  /* OF DO WHILE */                                                | 13962 INPUT_QUADS
| END INPUT_QUADS;                                                           | 13970 INPUT_QUADS
| /***************************************************************************/| 13976
| /*                       ***MAIN PROGRAM***                              */| 13976
| /*                                                                       */| 13976
| /*     THE MAIN PROGRAM FIRST CALLS THE PROCEDURE 'INPUT_QUADS' TO READ IN */| 13976
| /*AND ECHO-PRINT THE INPUT QUADS. THEN CALLS THE PROCEDURE 'FIND_INDX' AND */| 13976
| /*PROCEDURE 'DRIVER' TO PROCESS EACH QUAD AND CONVERT IT INTO VAX ASSEMBLY */| 13976
| /*LANGUAGE.                                                              */| 13976
| /***************************************************************************/| 13976
|     CALL INPUT_QUADS;  /* READ AND STORE MAXIMUM HUNDRED QUADS */          | 13976
|     /*                                                                     | 13984
|        PROCESS STORED QUADS                             */                 | 13984
|     CALL FIND_INDX(#OF_QUADS);                                             | 13984
|     CALL DRIVER(#OF_QUADS);                                                | 14000
|     /*                                                                     | 14016
|        OUTPUT ASSEMBLY VERSION OF QUADS                 */                 | 14016
|     CALL OUTPUT_VASM(0,#OF_GVINS-1,0);                                     | 14016
|     CALL OUTPUT_VASM(DECL#,VBUFSIZE,#OF_GVINS);                            | 14048
| EOF                                                                        | 14080
```

```
*  FILE CONTROL BLOCK  26000    13000    2    1    13000    1088    7008
*  LOAD FILE WRITTEN.
END OF COMPILATION SEPTEMBER 1, 1980.  CLOCK TIME = 18:50:54.91.

1016 CARDS CONTAINING 421 STATEMENTS WERE COMPILED.
NO ERRORS WERE DETECTED.
14088 BYTES OF PROGRAM, 3005 OF DATA, 3272 OF DESCRIPTORS, 729 OF STRINGS.  TOTAL CORE REQUIREMENT 21094 BYTES.


SYMBOL  TABLE  DUMP


#OF_DEFSYMB      : FIXED     AT 1788(11),   DECLARED ON LINE 41 AND REFERENCED 5 TIMES.
#OF_GVINS        : FIXED     AT 1772(11),   DECLARED ON LINE 27 AND REFERENCED 39 TIMES.
#OF_QUADS        : FIXED     AT 1400(11),   DECLARED ON LINE 14 AND REFERENCED 11 TIMES.
AFT_INDX         : BIT(8)    AT 1783(11),   DECLARED ON LINE 40 AND REFERENCED 3 TIMES.
ALLOCATION       : LABEL     AT 8794(14),   DECLARED ON LINE 598 AND REFERENCED 4 TIMES.
  PARAMETER  1   : CHARACTER AT 2984(13),   DECLARED ON LINE 599 AND REFERENCED 1 TIMES.
  PARAMETER  2   : CHARACTER AT 2988(13),   DECLARED ON LINE 599 AND REFERENCED 2 TIMES.
ARITH_OP         : LABEL     AT 11698(14),  DECLARED ON LINE 804 AND REFERENCED 1 TIMES.
  PARAMETER  1   : BIT(8)    AT 2776(11),   DECLARED ON LINE 805 AND REFERENCED 1 TIMES.
BLANK            : CHARACTER PROCEDURE AT 2134(14),   DECLARED ON LINE 138 AND REFERENCED 3 TIMES.
  PARAMETER  1   : CHARACTER AT 2472(13),   DECLARED ON LINE 139 AND REFERENCED 3 TIMES.
  PARAMETER  2   : FIXED     AT 1892(11),   DECLARED ON LINE 139 AND REFERENCED 2 TIMES.
CHOOS_BRTYP      : LABEL     AT 3882(14),   DECLARED ON LINE 332 AND REFERENCED 2 TIMES.
  PARAMETER  1   : BIT(8)    AT 2044(11),   DECLARED ON LINE 333 AND REFERENCED 2 TIMES.
  PARAMETER  2   : BIT(8)    AT 2045(11),   DECLARED ON LINE 333 AND REFERENCED 1 TIMES.
COMP_CHKBR       : LABEL     AT 5882(14),   DECLARED ON LINE 460 AND REFERENCED 6 TIMES.
  PARAMETER  1   : BIT(8)    AT 2225(11),   DECLARED ON LINE 461 AND REFERENCED 2 TIMES.
CONV_DATA        : LABEL     AT 3258(14),   DECLARED ON LINE 264 AND REFERENCED 5 TIMES.
  PARAMETER  1   : CHARACTER AT 2544(13),   DECLARED ON LINE 265 AND REFERENCED 1 TIMES.
  PARAMETER  2   : CHARACTER AT 2548(13),   DECLARED ON LINE 265 AND REFERENCED 1 TIMES.
  PARAMETER  3   : CHARACTER AT 2552(13),   DECLARED ON LINE 265 AND REFERENCED 1 TIMES.
CONVERT_TO_INTEG: LABEL      AT 2244(14),   DECLARED ON LINE 156 AND REFERENCED 4 TIMES.
  PARAMETER  1   : CHARACTER AT 2480(13),   DECLARED ON LINE 157 AND REFERENCED 3 TIMES.
CTYPE            : CHARACTER AT 2312(13),   DECLARED ON LINE 35 AND REFERENCED 2 TIMES.
DECL#            : FIXED     AT 1776(11),   DECLARED ON LINE 29 AND REFERENCED 8 TIMES.
DECLARATION      : LABEL     AT 11814(14),  DECLARED ON LINE 826 AND REFERENCED 1 TIMES.
  PARAMETER  1   : BIT(8)    AT 2796(11),   DECLARED ON LINE 827 AND REFERENCED 0 TIMES.
DIR_BRNCH        : LABEL     AT 12102(14),  DECLARED ON LINE 848 AND REFERENCED 1 TIMES.
  PARAMETER  1   : BIT(8)    AT 2824(11),   DECLARED ON LINE 849 AND REFERENCED 1 TIMES.
DRIVER           : LABEL     AT 12892(14),  DECLARED ON LINE 922 AND REFERENCED 1 TIMES.
  PARAMETER  1   : FIXED     AT 2908(11),   DECLARED ON LINE 923 AND REFERENCED 1 TIMES.
FBRDESQ          : FIXED     AT 1404(11),   DECLARED ON LINE 15 AND REFERENCED 2 TIMES.
FBRDESQ_IND      : BIT(8)    AT 1444(11),   DECLARED ON LINE 16 AND REFERENCED 5 TIMES.
FBRLABL_GEN      : LABEL     AT 12262(14),  DECLARED ON LINE 879 AND REFERENCED 1 TIMES.
  PARAMETER  1   : FIXED     AT 2852(11),   DECLARED ON LINE 880 AND REFERENCED 1 TIMES.
FIND_INDX        : LABEL     AT 13332(14),  DECLARED ON LINE 956 AND REFERENCED 1 TIMES.
  PARAMETER  1   : FIXED     AT 2952(11),   DECLARED ON LINE 957 AND REFERENCED 1 TIMES.
FLABL#           : BIT(8)    AT 1781(11),   DECLARED ON LINE 38 AND REFERENCED 4 TIMES.
GEN_REG#         : CHARACTER AT 2396(13),   DECLARED ON LINE 36 AND REFERENCED 6 TIMES.
GLABL            : CHARACTER PROCEDURE AT 3380(14),   DECLARED ON LINE 285 AND REFERENCED 3 TIMES.
  PARAMETER  1   : CHARACTER AT 2568(13),   DECLARED ON LINE 286 AND REFERENCED 3 TIMES.
IDENT_OPX        : LABEL     AT 2920(14),   DECLARED ON LINE 235 AND REFERENCED 14 TIMES.
  PARAMETER  1   : BIT(8)    AT 1968(11),   DECLARED ON LINE 236 AND REFERENCED 2 TIMES.
IDENT_UTOPX      : LABEL     AT 2666(14),   DECLARED ON LINE 198 AND REFERENCED 1 TIMES.
  PARAMETER  1   : BIT(8)    AT 1948(11),   DECLARED ON LINE 199 AND REFERENCED 1 TIMES.
INDX_REG         : CHARACTER AT 2436(13),   DECLARED ON LINE 39 AND REFERENCED 5 TIMES.
INDX_SYMB        : CHARACTER AT 2420(13),   DECLARED ON LINE 39 AND REFERENCED 2 TIMES.
INDX#            : BIT(8)    AT 1784(11),   DECLARED ON LINE 40 AND REFERENCED 5 TIMES.
```

```
LABL#              : BIT(8)     AT 1780(11),    DECLARED ON LINE 37 AND REFERENCED 3 TIMES.
LOGIC_OP           : LABEL      AT 10712(14),   DECLARED ON LINE 742 AND REFERENCED 1 TIMES.
  PARAMETER   1    : BIT(8)     AT 2668(11),    DECLARED ON LINE 743 AND REFERENCED 7 TIMES.
MISC               : LABEL      AT 10282(14),   DECLARED ON LINE 697 AND REFERENCED 1 TIMES.
  PARAMETER   1    : FIXED      AT 2608(11),    DECLARED ON LINE 698 AND REFERENCED 1 TIMES.
MOVE_CONV          : LABEL      AT 10608(14),   DECLARED ON LINE 730 AND REFERENCED 1 TIMES.
  PARAMETER   1    : BIT(8)     AT 2648(11),    DECLARED ON LINE 731 AND REFERENCED 1 TIMES.
NAME               : CHARACTER  AT 2228(13),    DECLARED ON LINE 35 AND REFERENCED 2 TIMES.
NVAL_OF_QID        : BIT(8)     AT 1398(11),    DECLARED ON LINE 13 AND REFERENCED 6 TIMES.
OPND               : CHARACTER  AT 2416(13),    DECLARED ON LINE 36 AND REFERENCED 29 TIMES.
OPXCT              : CHARACTER  AT 2412(13),    DECLARED ON LINE 36 AND REFERENCED 14 TIMES.
OP1CT              : CHARACTER  AT 2400(13),    DECLARED ON LINE 36 AND REFERENCED 25 TIMES.
OP2CT              : CHARACTER  AT 2404(13),    DECLARED ON LINE 36 AND REFERENCED 33 TIMES.
OP3CT              : CHARACTER  AT 2408(13),    DECLARED ON LINE 36 AND REFERENCED 30 TIMES.
OUTPUT_VASM        : LABEL      AT 12564(14),   DECLARED ON LINE 899 AND REFERENCED 2 TIMES.
  PARAMETER   1    : FIXED      AT 2888(11),    DECLARED ON LINE 900 AND REFERENCED 1 TIMES.
  PARAMETER   2    : FIXED      AT 2892(11),    DECLARED ON LINE 900 AND REFERENCED 1 TIMES.
  PARAMETER   3    : FIXED      AT 2896(11),    DECLARED ON LINE 900 AND REFERENCED 3 TIMES.
PROC_ARITH         : LABEL      AT 7088(14),    DECLARED ON LINE 506 AND REFERENCED 3 TIMES.
  PARAMETER   1    : CHARACTER  AT 2848(13),    DECLARED ON LINE 507 AND REFERENCED 2 TIMES.
PROC_BG            : LABEL      AT 9134(14),    DECLARED ON LINE 629 AND REFERENCED 1 TIMES.
PROC_BR            : LABEL      AT 10000(14),   DECLARED ON LINE 664 AND REFERENCED 1 TIMES.
PROC_INDX          : LABEL      AT 4558(14),    DECLARED ON LINE 368 AND REFERENCED 1 TIMES.
PROC_MOV           : LABEL      AT 5048(14),    DECLARED ON LINE 402 AND REFERENCED 1 TIMES.
PROC_UNST          : LABEL      AT 5700(14),    DECLARED ON LINE 434 AND REFERENCED 1 TIMES.
QBRFLG             : BIT(8)     AT 1340(11),    DECLARED ON LINE 7 AND REFERENCED 2 TIMES.
QIDBUF             : CHARACTER  AT 1044(13),    DECLARED ON LINE 6 AND REFERENCED 10 TIMES.
QNJM               : FIXED      AT 1392(11),    DECLARED ON LINE 12 AND REFERENCED 65 TIMES.
QOP1               : CHARACTER  AT 228(13),     DECLARED ON LINE 6 AND REFERENCED 14 TIMES.
QOP2               : CHARACTER  AT 432(13),     DECLARED ON LINE 6 AND REFERENCED 11 TIMES.
QOP3               : CHARACTER  AT 636(13),     DECLARED ON LINE 6 AND REFERENCED 6 TIMES.
QUAD_GROUP         : BIT(8)     AT 1396(11),    DECLARED ON LINE 13 AND REFERENCED 2 TIMES.
QUAD#_IN_GROUP     : BIT(8)     AT 1397(11),    DECLARED ON LINE 13 AND REFERENCED 8 TIMES.
QUADNAME           : CHARACTER  AT 24(13),      DECLARED ON LINE 6 AND REFERENCED 1 TIMES.
QUADTYPE           : CHARACTER  AT 840(13),     DECLARED ON LINE 6 AND REFERENCED 1 TIMES.
SEARCH_TYPE        : CHARACTER PROCEDURE AT 1364(14),   DECLARED ON LINE 66 AND REFERENCED 1 TIMES.
  PARAMETER   1    : CHARACTER  AT 2448(13),    DECLARED ON LINE 67 AND REFERENCED 1 TIMES.
SHIFT_OP           : LABEL      AT 11088(14),   DECLARED ON LINE 775 AND REFERENCED 1 TIMES.
  PARAMETER   1    : BIT(8)     AT 2720(11),    DECLARED ON LINE 776 AND REFERENCED 0 TIMES.
SKIP_NEXT_QUAD     : BIT(8)     AT 1445(11),    DECLARED ON LINE 17 AND REFERENCED 6 TIMES.
SRCH_FBRDESQ       : LABEL      AT 10078(14),   DECLARED ON LINE 679 AND REFERENCED 1 TIMES.
  PARAMETER   1    : FIXED      AT 2596(11),    DECLARED ON LINE 681 AND REFERENCED 1 TIMES.
SRCH_INDX          : CHARACTER PROCEDURE AT 2430(14),   DECLARED ON LINE 175 AND REFERENCED 2 TIMES.
  PARAMETER   1    : CHARACTER  AT 2484(13),    DECLARED ON LINE 176 AND REFERENCED 1 TIMES.
  PARAMETER   2    : BIT(8)     AT 1925(11),    DECLARED ON LINE 176 AND REFERENCED 2 TIMES.
SRCH_QNUM          : LABEL      AT 1478(14),    DECLARED ON LINE 83 AND REFERENCED 1 TIMES.
  PARAMETER   1    : FIXED      AT 1844(11),    DECLARED ON LINE 84 AND REFERENCED 3 TIMES.
STOR_VAX_INS       : LABEL      AT 1290(14),    DECLARED ON LINE 50 AND REFERENCED 27 TIMES.
  PARAMETER   1    : CHARACTER  AT 2440(13),    DECLARED ON LINE 51 AND REFERENCED 1 TIMES.
  PARAMETER   2    : CHARACTER  AT 2444(13),    DECLARED ON LINE 51 AND REFERENCED 1 TIMES.
  PARAMETER   3    : FIXED      AT 1796(11),    DECLARED ON LINE 51 AND REFERENCED 1 TIMES.
  PARAMETER   4    : FIXED      AT 1800(11),    DECLARED ON LINE 51 AND REFERENCED 4 TIMES.
SUBENT_ADDR        : LABEL      AT 8648(14),    DECLARED ON LINE 576 AND REFERENCED 1 TIMES.
SYMBTAYL           : CHARACTER PROCEDURE AT 1732(14),   DECLARED ON LINE 116 AND REFERENCED 15 TIMES.
  PARAMETER   1    : CHARACTER  AT 2452(13),    DECLARED ON LINE 117 AND REFERENCED 8 TIMES.
VLABEL_FLD         : CHARACTER  AT 1248(13),    DECLARED ON LINE 22 AND REFERENCED 9 TIMES.
VOPERAT            : CHARACTER  AT 2220(13),    DECLARED ON LINE 28 AND REFERENCED 14 TIMES.
VOPERAT_FLD        : CHARACTER  AT 1572(13),    DECLARED ON LINE 22 AND REFERENCED 4 TIMES.
VOPRND             : CHARACTER  AT 2224(13),    DECLARED ON LINE 28 AND REFERENCED 12 TIMES.
VOPRND_FLD         : CHARACTER  AT 1896(13),    DECLARED ON LINE 22 AND REFERENCED 4 TIMES.
VQNUM              : FIXED      AT 1448(11),    DECLARED ON LINE 23 AND REFERENCED 6 TIMES.
```

```
MACRO DEFINITIONS:

DIGIT                  LITERALLY: 0123456789
DX_SIZE                LITERALLY: 500
QBUFSIZE               LITERALLY: 50
VBUFSIZE               LITERALLY: 80
MAX#_OF_FBR            LITERALLY: 9
MAX#_OF_SYMBS          LITERALLY: 20

IDCOMPARES          = 42852
SYMBOL TABLE SIZE = 143
MACRO DEFINITIONS = 6
STACKING DECISIONS= 18226
SCAN                = 4834
EMITRR              = 548
EMITRX              = 3438
FORCFACCUMULATOR    = 1356
ARITHEMIT           = 230
GENSTORE            = 243
FIXBFW              = 262
FIXDATAWORD         = 12
FIXCHW              = 435
GETDATA             = 0
GETCODE             = 3
FINDADDRESS         = 698
SHORTCFIX           = 429
LONGCFIX            = 6
SHORTDFIX           = 12
LONGDFIX            = 0
FREE STRING AREA    = 111222

REGISTER VALUES (RELATIVE TO R11):
R4 = 0
R5 = 0
R6 = 0
R7 = 0
R8 = 0
R9 = 0
R10 = 0
R11 = 0
R12 = 0
R13 = 3000

 INSTRUCTION FREQUENCIES:

BALR     28
BCTR     3
BCR      56
LPR      1
LTR      8
LCR      1
NR       19
OR       12
XR       69
LR       180
CR       7
AR       22
```

[illegible]	[illegible]
LA	256
STC	81
IC	97
EX	62
BAL	201
BC	408
LH	6
ST	593
N	33
O	18
L	1058
C	66
A	58
S	31
M	2
D	5
AL	2
SL	2
SRL	65
SLL	94
SRA	60
SRDA	5
STM	28
TM	3
OI	1
LM	29

```
TOTAL TIME IN COMPILER    0:1:49.22.
SET UP TIME               0:0:7.21.
ACTUAL COMPILATION TIME   0:1:34.99.
POST-COMPILATION TIME     0:0:7.02.
COMPILATION RATE: 641 CARDS PER MINUTE.
```

Appendix A2

Quadruple To VAX 11/780

Microcode Translator

with Example

```
| /**********************************************************************/|  1286
| /*                         DECLARATION                                */|  1286
| /**********************************************************************/|  1286
|   DECLARE (QUADNAME,QUADOPD1,QUADOPD2,QUADOPD3)(33) CHARACTER;           |  1286
|   DECLARE (MICRO_LABL_FLD,MICRO_OPRT_FLD)(70) CHARACTER;                 |  1286
|   DECLARE (OPND,QOPND)(20) CHARACTER;                                    |  1286
|   DECLARE #OF_LINE FIXED INITIAL(0), IN# FIXED INITIAL(0),              |  1286
|           INDX FIXED INITIAL(0), #OF_QUADS FIXED INITIAL(0),           |  1286
|           (QNUM,NVAL_OF_QID) BIT(8),                                    |  1286
|           LABL_FLD_INDX(33) BIT (8),                                     |  1286
|           FBRDESQ(9) BIT(8),                                             |  1286
|           QBRFLG(33) BIT(1),                                             |  1286
|           INDUM BIT(8) INITIAL(0),                                       |  1286
|           SKIP_NEXT_QUAD BIT(1) INITIAL(0);                              |  1286
|   DECLARE (ALPHA,DIGIT) CHARACTER;                                       |  1286
|   /*                                                                     |  1286
|   DECLARE DIGIT LITERALLY '0123456789';                                  |  1286
|   DECLARE ALPHA LITERALLY 'ABCEDFGHIJKLMNOPQRSTUVWXYZ';                  |  1286
|   */                                                                     |  1286
|   DECLARE QUAD_TAB(77) CHARACTER INITIAL(                                |  1286
|           'BR  ','','','','BG  ','','','BRT ','BRF ','',                 |  1286
|           '','','','','','','','','','',                                 |  1286
|           'BYTE','BOOL','FULL','HALF','ADDR','EQU','','','','',          |  1286
|           'ADD ',' SUB ','','','','','','','','',                        |  1286
|           'SHLA','SHRA','','','','','','','','',                         |  1286
|           '','','','','','','NE','LT','GT  ','',                         |  1286
|           'MOV ','','INDX','','','','','','','',                         |  1286
|           '','','','','','RETN','','UNST');                              |  1286
|   DECLARE CONST_ARRAY(56) CHARACTER INITIAL(                             |  1286
|           '8','1','2','3','4','20','160','52','40','64','80','12288','239', |  1286
|           '128','-32768','255','-256','30','63','127','7','15','16','-24', |  1286
|           '-16','-8','32','48','24','1023','12','13','31','7936','176','0', |  1286
|           '124','-32','96','-17','-15','25','-7','-1','136','240','192', |  1286
|           '6','9','-10','-11','26','36','27','-4','10','126');           |  1286
| /**********************************************************************/|  1286
| /*                      PROCEDURE IDENT_QNAME                         */|  1286
| /**********************************************************************/|  1286
| IDENT_QNAME:PROCEDURE(S) BIT(8);                                         |  1286
|   DECLARE S CHARACTER, INX BIT(8), FOUND BIT(1);                         |  1286 IDENT_QNAME
|     INX=0;                                                               |  1294 IDENT_QNAME
|     FOUND=0;                                                             |  1300 IDENT_QNAME
|     DO WHILE (INX<=77)&(FOUND=0);                                        |  1306 IDENT_QNAME
|        IF S=QUAD_TAB(INX) THEN FOUND=1;                                  |  1360 IDENT_QNAME C7 = 77.
|           ELSE INX=INX+1;                                                |  1412 IDENT_QNAME
|     END;  /* OF DO WHILE */                                              |  1430 IDENT_QNAME
|     IF INX¬=78 THEN RETURN INX;                                          |  1434 IDENT_QNAME
|        ELSE OUTPUT='ILLEGAL QUAD HAS BEEN READ IN.';                     |  1460 IDENT_QNAME C8 = 78.
| END IDENT_QNAME;                                                         |  1484 IDENT_QNAME
| /**********************************************************************/|  1490
| /*                      PROCEDURE IDENT_CONST                         */|  1490
| /**********************************************************************/|  1490
| IDENT_CONST:PROCEDURE(OPX) BIT(8);                                       |  1490
|   DECLARE I BIT(8), FOUND BIT(1), OPX CHARACTER;                         |  1490 IDENT_CONST
|     I=0;                                                                 |  1498 IDENT_CONST
|     FOUND=0;                                                             |  1504 IDENT_CONST
|     DO WHILE (I<=55)&(FOUND=0);                                          |  1510 IDENT_CONST
```

```
      IF OPX=CONST_ARRAY(I) THEN FOUND=1;                                    1564 IDENT_CONST C9 = 55.
         ELSE I=I+1;                                                         1616 IDENT_CONST
   END;  /* OF DO WHILE */                                                   1634 IDENT_CONST
   RETURN FOUND;                                                             1638 IDENT_CONST
END IDENT_CONST;                                                             1650 IDENT_CONST
/************************************************************************/   1656
/*                        PROCEDURE SYMBTAYL                             */   1656
/************************************************************************/   1656
SYMBTAYL:PROCEDURE(OPX) CHARACTER;                                           1656
  DECLARE (OPX,SYMBOL) CHARACTER, (QOPLENG,I) FIXED;                         1656 SYMBTAYL
   QOPLENG=LENGTH(OPX);                                                      1664 SYMBTAYL
   SYMBOL='';                                                                1686 SYMBTAYL
   IF SUBSTR(OPX,0,1)¬=' ' THEN SYMBOL=SUBSTR(OPX,0,1);                      1692 SYMBTAYL
   IF SUBSTR(OPX,1,1)¬='0' THEN SYMBOL=SYMBOL||SUBSTR(OPX,1,1);              1750 SYMBTAYL
   I=2;                                                                      1824 SYMBTAYL
   DO WHILE (BYTE(OPX,I)=BYTE('0'))&(I<11);                                  1832 SYMBTAYL
      I=I+1;                                                                 1890 SYMBTAYL C10 = 240. C11 = 11.
   END;  /* OF DO WHILE */                                                   1902 SYMBTAYL
   IF (BYTE(OPX,10)=BYTE('0'))&(I=11) THEN                                   1936 SYMBTAYL
      SYMBOL=SYMBOL||'0';                                                    1956 SYMBTAYL
   ELSE SYMBOL=SYMBOL||SUBSTR(OPX,I,QOPLENG-I);                              1988 SYMBTAYL
   RETURN SYMBOL;                                                            2042 SYMBTAYL
END SYMBTAYL;                                                                2052 SYMBTAYL
/************************************************************************/   2058
/*                        PROCEDURE COMENT_LINE                          */   2058
/************************************************************************/   2058
COMENT_LINE:PROCEDURE;                                                       2058
  #OF_LINE=#OF_LINE+1;                                                       2058 COMENT_LINE
  MICRO_OPRT_FLD(#OF_LINE)=';----------------------;';                       2078 COMENT_LINE
  #OF_LINE=#OF_LINE+1;                                                       2094 COMENT_LINE
END COMENT_LINE;                                                             2106 COMENT_LINE
/************************************************************************/   2112
/*                        PROCEDURE CONVERT_TO_INTEG                     */   2112
/************************************************************************/   2112
CONVERT_TO_INTEG:PROCEDURE(S) BIT(8);                                        2112
  DECLARE S CHARACTER, (I,NVAL) BIT(8);                                      2112 CONVERT_TO_INTEG
   NVAL=0;                                                                   2120 CONVERT_TO_INTEG
   DO I=0 TO LENGTH(S)-1;                                                    2126 CONVERT_TO_INTEG
   NVAL=NVAL*10+BYTE(S,I)-BYTE('0');                                         2180 CONVERT_TO_INTEG
   END;  /* OF DO STATEMENT */                                               2214 CONVERT_TO_INTEG C12 = 10.
   RETURN NVAL;                                                              2218 CONVERT_TO_INTEG
END CONVERT_TO_INTEG;                                                        2230 CONVERT_TO_INTEG
/************************************************************************/   2236
/*                        PROCEDURE FIND_REG#                            */   2236
/************************************************************************/   2236
FIND_REG#:PROCEDURE(OPX) CHARACTER;                                          2236
  DECLARE (REGNUM,OPX) CHARACTER, IJ BIT(8), FOUND BIT(8);                   2236 FIND_REG#
   IJ=0;                                                                     2244 FIND_REG#
   FOUND=0;                                                                  2250 FIND_REG#
   DO WHILE (IJ<=INDX-1)&(FOUND=0);                                          2256 FIND_REG#
      IF OPX=QOPND(IJ) THEN                                                  2316 FIND_REG#
         DO;                                                                 2368 FIND_REG#
         FOUND=1;                                                            2360 FIND_REG#
         REGNUM=OPND(IJ);                                                    2368 FIND_REG#
         END;  /* OF IF THEN DO */                                           2386 FIND_REG#
         ELSE IJ=IJ+1;                                                       2386 FIND_REG#
   END;  /* OF DO WHILE */                                                   2404 FIND_REG#
   RETURN REGNUM;                                                            2408 FIND_REG#
END FIND_REG#;                                                               2418 FIND_REG#
/************************************************************************/   2424
/*                        PROCEDURE PROC_MOV                             */   2424
/************************************************************************/   2424
PROC_MOV:PROCEDURE;                                                          2424
  DECLARE (OP1,OP3,MOP1,MOP3) CHARACTER, BLIN_CONST BIT(1);                  2424 PROC_MOV
```

```
    LABL_FLD_INDX(QNUM)=#OF_LINE;                                       | 2484 PROC_MOV
    IF (SUBSTR(OP1,0,1)='#')&(SUBSTR(OP3,0,1)='U') THEN                 | 2498 PROC_MOV
       DO;                                                              | 2604 PROC_MOV
       MOP3=FIND_REG#(OP3);                                             | 2612 PROC_MOV
       BLIN_CONST=IDENT_CONST(SUBSTR(OP1,1));                           | 2628 PROC_MOV
       IF BLIN_CONST=1 THEN                                             | 2654 PROC_MOV C13 = 16777215.
          DO;                                                           | 2676 PROC_MOV
          IF OP1='#0' THEN MICRO_OPRT_FLD(#OF_LINE)='R(.'||MOP3||       | 2668 PROC_MOV
                                                 '.)_K(..ZERO.);';      | 2734 PROC_MOV
             ELSE MICRO_OPRT_FLD(#OF_LINE)='R(.'||MOP3||'.)_K(..'||     | 2766 PROC_MOV
                                          SUBSTR(OP1,1)||'.);';         | 2826 PROC_MOV
          CALL COMENT_LINE;                                             | 2892 PROC_MOV
          END;  /* OF IF THEN DO */                                     | 2896 PROC_MOV
          ELSE OUTPUT='NON-BLIN-CONSTANT NOT IMPLEMENTED YET';          | 2896 PROC_MOV
       END;  /* OF IF THEN DO */                                        | 2920 PROC_MOV
       ELSE DO;                                                         | 2920 PROC_MOV
       IF (SUBSTR(OP1,0,1)='U')&(SUBSTR(OP3,0,1)='U') THEN              | 2924 PROC_MOV
          DO;                                                           | 3030 PROC_MOV
          MOP1=FIND_REG#(OP1);                                          | 3038 PROC_MOV
          MOP3=FIND_REG#(OP3);                                          | 3054 PROC_MOV
          MICRO_OPRT_FLD(#OF_LINE)='LAB_R(.'||MOP1||'.);';              | 3070 PROC_MOV
          CALL COMENT_LINE;                                             | 3134 PROC_MOV
          MICRO_OPRT_FLD(#OF_LINE)='R(.'||MOP3||'.)_LA;';               | 3138 PROC_MOV
          CALL COMENT_LINE;                                             | 3202 PROC_MOV
          END;  /* OF IF THEN DO */                                     | 3206 PROC_MOV
          ELSE OUTPUT='THIS TYPE OF MOVE HAS NOT BEEN IMPLEMENTED YET'; | 3206 PROC_MOV
       END;  /* OF ELSE DO */                                           | 3230 PROC_MOV
END PROC_MOV;                                                           | 3230 PROC_MOV
/*********************************************************************/| 3236
/*                       PROCEDURE PROC_INDX                         */| 3236
/*********************************************************************/| 3236
PROC_INDX:PROCEDURE;                                                    | 3236
  OUTPUT='INDX QUAD HAS NOT BEEN IMPLEMENTED YET';                      | 3236 PROC_INDX
END PROC_INDX;                                                          | 3264 PROC_INDX
/*********************************************************************/| 3270
/*                       PROCEDURE MOV_CONV                          */| 3270
/*********************************************************************/| 3270
MOV_CONV:PROCEDURE(QINGRP);                                             | 3270
  DECLARE QINGRP BIT(8);                                                | 3270 MOV_CONV
    DO CASE QINGRP;                                                     | 3278 MOV_CONV
       CALL PROC_MOV;                                                   | 3300 MOV_CONV
       OUTPUT='QUAD NUMBER IS 61, NOT IMPLEMENTED YET';                 | 3304 MOV_CONV CASE 0.
       CALL PROC_INDX;                                                  | 3328 MOV_CONV CASE 1.
    END;  /* OF DO CASE */                                              | 3336 MOV_CONV CASE 2.
END MOV_CONV;                                                           | 3340 MOV_CONV
/*********************************************************************/| 3346
/*                       PROCEDURE ARITH_OP                          */| 3346
/*********************************************************************/| 3346
ARITH_OP:PROCEDURE(I);                                                  | 3346
  DECLARE I BIT(8), (OP1,OP2,OP3,MOP1,MOP2,MOP3) CHARACTER,             | 3346 ARITH_OP
          BLIN_CONST BIT(1);                                            | 3354 ARITH_OP
    OP1=SYMBTAYL(QUADOPD1(QNUM));                                       | 3354 ARITH_OP
    OP2=SYMBTAYL(QUADOPD2(QNUM));                                       | 3380 ARITH_OP
    OP3=SYMBTAYL(QUADOPD3(QNUM));                                       | 3406 ARITH_OP
    LABL_FLD_INDX(QNUM)=#OF_LINE;                                       | 3432 ARITH_OP
    IF I=0 THEN                                                         | 3446 ARITH_OP
       IF (SUBSTR(OP1,0,1)='U')&(SUBSTR(OP2,0,1)='U')&                  | 3468 ARITH_OP
                        (SUBSTR(OP3,0,1)='U') THEN                      | 3566 ARITH_OP
          DO;                                                           | 3622 ARITH_OP
          MOP1=FIND_REG#(OP1);                                          | 3630 ARITH_OP
          MOP2=FIND_REG#(OP2);                                          | 3646 ARITH_OP
          MOP3=FIND_REG#(OP3);                                          | 3662 ARITH_OP
          MICRO_OPRT_FLD(#OF_LINE)='Q_R(.'||MOP1||'.);';                | 3678 ARITH_OP
          CALL COMENT_LINE;                                             | 3742 ARITH_OP
          [illegible]                                                   | 3746 ARITH_OP
```

```
          MICRO_OPRT_FLD(#OF_LINE)='R(.'||MOP3||'.)_ALU, ALU_LA+Q;';               | 3814 ARITH_OP
          CALL COMENT_LINE;                                                        | 3878 ARITH_OP
          END;  /* OF IF THEN DO */                                                | 3882 ARITH_OP
          ELSE DO;                                                                 | 3882 ARITH_OP
          IF (SUBSTR(OP1,0,1)='#')&(SUBSTR(OP3,0,1)='U') THEN                      | 3886 ARITH_OP
             DO;                                                                   | 3992 ARITH_OP
             MOP3=FIND_REG#(OP3);                                                  | 4000 ARITH_OP
             BLIN_CONST=IDENT_CONST(SUBSTR(OP1,1));                                | 4016 ARITH_OP
             IF BLIN_CONST=1 THEN                                                  | 4042 ARITH_OP
                DO;                                                                | 4064 ARITH_OP
                MICRO_OPRT_FLD(#OF_LINE)='LAB_R(.'||MOP3||'.);';                   | 4056 ARITH_OP
                CALL COMENT_LINE;                                                  | 4120 ARITH_OP
                MICRO_OPRT_FLD(#OF_LINE)='ALU_LA+K(..'||SUBSTR(OP1,1)||'..),';     | 4124 ARITH_OP
                #OF_LINE=#OF_LINE+1;                                               | 4202 ARITH_OP
                MICRO_OPRT_FLD(#OF_LINE)='R(.'||MOP3||'.)_ALU;';                   | 4214 ARITH_OP
                CALL COMENT_LINE;                                                  | 4278 ARITH_OP
                END;  /* OF IF THEN DO  */                                         | 4282 ARITH_OP
                ELSE OUTPUT='NON-BLIN-CONSTANT HAS NOT IMPLEMENTED YET';           | 4282 ARITH_OP
             END;  /* OF IF THEN DO */                                             | 4310 ARITH_OP
             ELSE OUTPUT='SUCH TYPE OF ADD QUAD NOT IMPLEMENTED YET';              | 4310 ARITH_OP
          END;  /* OF ELSE DO */                                                   | 4338 ARITH_OP
    IF I¬=0 THEN                                                                   | 4338 ARITH_OP
       OUTPUT='NOT IMPLEMENTED YET';                                               | 4364 ARITH_OP
END ARITH_OP;                                                                      | 4376 ARITH_OP
/*************************************************************************/| 4382
/*                        PROCEDURE GLABL                                */| 4382
/*************************************************************************/| 4382
GLABL:PROCEDURE(OPX);                                                              | 4382
  DECLARE VOPD BIT(8), OPX CHARACTER;                                              | 4382 GLABL
    VOPD=CONVERT_TO_INTEG(OPX);                                                    | 4394 GLABL
    IF VOPD<=QNUM THEN                                                             | 4410 GLABL
       DO;                                                                         | 4440 GLABL
       IF SUBSTR(MICRO_LABL_FLD(LABL_FLD_INDX(VOPD)),0,5)='LABL-' THEN             | 4432 GLABL
          MICRO_OPRT_FLD(#OF_LINE)='J/'||MICRO_LABL_FLD(LABL_FLD_INDX(VOPD))       | 4506 GLABL C14 = 67108864.
                                   ||';';                                          | 4508 GLABL
          ELSE DO;                                                                 | 4578 GLABL
          MICRO_OPRT_FLD(#OF_LINE)='J/LABL-'||SUBSTR(ALPHA,IN#,1)||';';            | 4586 GLABL
          MICRO_LABL_FLD(LABL_FLD_INDX(VOPD))='LABL-'||SUBSTR(ALPHA,IN#,1)         | 4662 GLABL
                                                  ||':';                           | 4682 GLABL
          IN#=IN#+1;                                                               | 4746 GLABL
          END;  /* OF ELSE DO */                                                   | 4758 GLABL
       END;  /* OF IF THEN DO */                                                   | 4758 GLABL
       ELSE DO;                                                                    | 4758 GLABL
       QBRFLG(VOPD)=1;                                                             | 4766 GLABL
       FBRDESQ(INDUM)=VOPD;                                                        | 4780 GLABL
       MICRO_OPRT_FLD(#OF_LINE)='J/LBLF-'||SUBSTR(ALPHA,INDUM,1)||';';             | 4796 GLABL
       INDUM=INDUM+1;                                                              | 4874 GLABL
       END;  /* OF ELSE DO */                                                      | 4888 GLABL
    CALL COMENT_LINE;                                                              | 4888 GLABL
END GLABL;                                                                         | 4892 GLABL
/*************************************************************************/| 4898
/*                        PROCEDURE PROC_BR                              */| 4898
/*************************************************************************/| 4898
PROC_BR:PROCEDURE;                                                                 | 4898
  DECLARE OP1 CHARACTER;                                                           | 4898 PROC_BR
    OP1=SYMBTAYL(QUADOPD1(QNUM));                                                  | 4910 PROC_BR
    LABL_FLD_INDX(QNUM)=#OF_LINE;                                                  | 4936 PROC_BR
    CALL GLABL(OP1);                                                               | 4950 PROC_BR
END PROC_BR;                                                                       | 4966 PROC_BR
/*************************************************************************/| 4972
/*                        PROCEDURE PROC_BG                              */| 4972
/*************************************************************************/| 4972
PROC_BG:PROCEDURE;                                                                 | 4972
  DECLARE (OP1,OP2,OP3,MOP2,MOP3) CHARACTER, VOPD BIT(8);                          | 4972 PROC_BG
        [illegible]                                                                | 4984 PROC_BG
```

```
      OP3=SYMBTAYL(QUADOPD3(QNUM));                                    | 5036 PROC_BG
      MOP2=FIND_REG#(OP2);                                             | 5062 PROC_BG
      MOP3=FIND_REG#(OP3);                                             | 5078 PROC_BG
      LABL_FLD_INDX(QNUM)=#OF_LINE;                                    | 5094 PROC_BG
      MICRO_OPRT_FLD(#OF_LINE)='Q_R(.'||MOP2||'.);';                   | 5108 PROC_BG
      CALL COMENT_LINE;                                                | 5172 PROC_BG
      MICRO_OPRT_FLD(#OF_LINE)='ALU_LA-Q,';                            | 5176 PROC_BG
      MICRO_OPRT_FLD(#OF_LINE+1)='LAB_R(.'||MOP3||'.),';               | 5192 PROC_BG
      MICRO_OPRT_FLD(#OF_LINE+2)='CLK.UBCC;';                          | 5260 PROC_BG
      #OF_LINE=#OF_LINE+2;                                             | 5280 PROC_BG
      CALL COMENT_LINE;                                                | 5292 PROC_BG
      MICRO_OPRT_FLD(#OF_LINE)='ALU?;';                                | 5296 PROC_BG
      #OF_LINE=#OF_LINE+1;                                             | 5312 PROC_BG
      MICRO_LABL_FLD(#OF_LINE)='=1110';                                | 5324 PROC_BG
      MICRO_OPRT_FLD(#OF_LINE)=';----------------------;';             | 5340 PROC_BG
      #OF_LINE=#OF_LINE+1;                                             | 5356 PROC_BG
      CALL GLABL(OP1);                                                 | 5368 PROC_BG
 END PROC_BG;                                                          | 5384 PROC_BG
 /**********************************************************************/| 5390
 /*                        PROCEDURE DIR_BRNCH                         */| 5390
 /**********************************************************************/| 5390
 DIR_BRNCH:PROCEDURE(QINGRP);                                          | 5390
   DECLARE QINGRP BIT(8);                                              | 5390 DIR_BRNCH
      DO CASE QINGRP;                                                  | 5402 DIR_BRNCH
         CALL PROC_BR;                                                 | 5428 DIR_BRNCH
         OUTPUT='NO PROC_BZ YET';                                      | 5436 DIR_BRNCH CASE 0.
         OUTPUT='NO PORC_BP YET';                                      | 5464 DIR_BRNCH CASE 1.
         OUTPUT='NO PROC_BN YET';                                      | 5492 DIR_BRNCH CASE 2.
         CALL PROC_BG;                                                 | 5520 DIR_BRNCH CASE 3.
      END;  /* OF DO CASE */                                           | 5536 DIR_BRNCH CASE 4.
 END DIR_BRNCH;                                                        | 5544 DIR_BRNCH
 /**********************************************************************/| 5550
 /*                        PROCEDURE PROC_UNST                         */| 5550
 /**********************************************************************/| 5550
 PROC_UNST:PROCEDURE;                                                  | 5550
   LABL_FLD_INDX(QNUM)=#OF_LINE;                                       | 5550 PROC_UNST
   IF IDENT_QNAME(QUADNAME(QNUM+1))=75 THEN                            | 5576 PROC_UNST
      DO;                                                              | 5622 PROC_UNST C15 = 75.
      SKIP_NEXT_QUAD=1;                                                | 5614 PROC_UNST
      MICRO_OPRT_FLD(#OF_LINE)='PC_PC+1, CLR.IB.OPC,';                 | 5622 PROC_UNST
      MICRO_OPRT_FLD(#OF_LINE+1)='J/062;';                             | 5638 PROC_UNST
      #OF_LINE=#OF_LINE+1;                                             | 5658 PROC_UNST
      CALL COMENT_LINE;                                                | 5670 PROC_UNST
      END;  /* OF IF THEN DO */                                        | 5674 PROC_UNST
      ELSE OUTPUT='NOT IMPLEMENTED YET';                               | 5674 PROC_UNST
 END PROC_UNST;                                                        | 5702 PROC_UNST
 /**********************************************************************/| 5708
 /*                        PROCEDURE PROC_RETN                         */| 5708
 /**********************************************************************/| 5708
 PROC_RETN:PROCEDURE;                                                  | 5708
   OUTPUT='NOT IMPLEMENTED YET';                                       | 5708 PROC_RETN
 END PROC_RETN;                                                        | 5740 PROC_RETN
 /**********************************************************************/| 5746
 /*                        PROCEDURE MISC                              */| 5746
 /**********************************************************************/| 5746
 MISC:PROCEDURE(QINGRP);                                               | 5746
   DECLARE QINGRP FIXED;                                               | 5746 MISC
      DO CASE QINGRP;                                                  | 5758 MISC
         OUTPUT='NOT IMPLEMENTED YET';                                 | 5782 MISC
         OUTPUT='NOT IMPLEMENTED YET';                                 | 5802 MISC CASE 0.
         OUTPUT='NOT IMPLEMENTED YET';                                 | 5830 MISC CASE 1.
         OUTPUT='NOT IMPLEMENTED YET';                                 | 5858 MISC CASE 2.
         OUTPUT='NOT IMPLEMENTED YET';                                 | 5886 MISC CASE 3.
         CALL PROC_RETN;                                               | 5914 MISC CASE 4.
```

```
      END;  /* OF DO CASE  */                                                     5974 MISC CASE 7.
END MISC;                                                                         5982 MISC
  /*******************************************************  ******************/  5988
/*                      PROCEDURE FBRLABL_GEN                                */  5988
/***************************************************************************/  5988
FBRLABL_GEN:PROCEDURE(FQUAD);                                                     5988
  DECLARE (FQUAD,I) BIT(8), FOUND BIT(1);                                         5988 FBRLABL_GEN
    I=0;                                                                          6000 FBRLABL_GEN
    FOUND=0;                                                                      6006 FBRLABL_GEN
    DO WHILE (FOUND=0)&(I<=INDUM-1);                                              6012 FBRLABL_GEN
       IF FBRDESQ(I)=FQUAD THEN FOUND=1;                                          6086 FBRLABL_GEN
          ELSE I=I+1;                                                             6122 FBRLABL_GEN
    END;  /* OF DO WHILE */                                                       6144 FBRLABL_GEN
    IF FOUND=1 THEN                                                               6152 FBRLABL_GEN
       MICRO_LABL_FLD(#OF_LINE)='LBLF-'||SUBSTR(ALPHA,I,1)||':';                  6178 FBRLABL_GEN
       ELSE OUTPUT='FORWARD BRANCH DETINATION QUD NOT FOUND';                     6248 FBRLABL_GEN
END FBRLABL_GEN;                                                                  6276 FBRLABL_GEN
/***************************************************************************/  6282
/*                      PROCEDURE MEM_LABL_GEN                               */  6282
/***************************************************************************/  6282
MEM_LABL_GEN:PROCEDURE;                                                           6282
  OUTPUT='MEM_LABL_GEN NOT BEEN IMPLEMENTED YET';                                 6282 MEM_LABL_GEN
END MEM_LABL_GEN;                                                                 6314 MEM_LABL_GEN
/***************************************************************************/  6320
/*                      PROCEDURE SHIFT_OP                                   */  6320
/***************************************************************************/  6320
SHIFT_OP:PROCEDURE(QINGRP);                                                       6320
  DECLARE QINGRP BIT(8);                                                          6320 SHIFT_OP
    OUTPUT='NOT IMPLEMENTED YET';                                                 6332 SHIFT_OP
END SHIFT_OP;                                                                     6352 SHIFT_OP
/***************************************************************************/  6358
/*                      PROCEDURE LOGIC_OP                                   */  6358
/***************************************************************************/  6358
LOGIC_OP:PROCEDURE(QINGRP);                                                       6358
  DECLARE QINGRP BIT(8);                                                          6358 LOGIC_OP
    OUTPUT='NOT IMPLEMENTED YET';                                                 6370 LOGIC_OP
END LOGIC_OP;                                                                     6390 LOGIC_OP
/***************************************************************************/  6396
/*                      PROCEDURE DECLARATION                                */  6396
/***************************************************************************/  6396
DECLARATION:PROCEDURE(QINGRP);                                                    6396
  DECLARE (NQGRP,NQINGRP,QINGRP) BIT(8), IND# BIT(8) INITIAL(0),                  6396 DECLARATION
          (REG#,SYMBOL) CHARACTER;                                                6408 DECLARATION
    NVAL_OF_QID=IDENT_QNAME(QUADNAME(QNUM+1));                                    6408 DECLARATION
    NQGRP=NVAL_OF_QID/10;                                                         6438 DECLARATION
    IF NQGRP=2 THEN                                                               6456 DECLARATION
       DO;                                                                        6482 DECLARATION C16 = 2.
       NQINGRP=NVAL_OF_QID MOD 10;                                                6474 DECLARATION
       IF NQINGRP=4 THEN                                                          6494 DECLARATION
          DO;                                                                     6520 DECLARATION C17 = 4.
          OUTPUT='';                                                              6512 DECLARATION
          MICRO_OPRT_FLD(#OF_LINE)='.TOC ''                 ''';                  6530 DECLARATION
          MICRO_OPRT_FLD(#OF_LINE+1)='.REGION    /1400,17FF';                     6546 DECLARATION
          MICRO_LABL_FLD(#OF_LINE+2)=';INPUT';                                    6566 DECLARATION
          #OF_LINE=#OF_LINE+3;                                                    6586 DECLARATION
          END;  /* OF IF THEN DO */                                               6598 DECLARATION
          ELSE DO;                                                                6598 DECLARATION
          SKIP_NEXT_QUAD=1;                                                       6606 DECLARATION
          REG#='R'||SUBSTR(DIGIT,IND#,1);                                         6614 DECLARATION
          IND#=IND#+1;                                                            6652 DECLARATION
          IF REG#='R8' THEN CALL MEM_LABL_GEN;                                    6666 DECLARATION
             ELSE DO;                                                             6716 DECLARATION
             SYMBOL=SYMBTAYL(QUADOPD1(QNUM));                                     6724 DECLARATION
             MICRO_LABL_FLD(#OF_LINE)=';';                                        6750 DECLARATION
```

```
            QPND(INDX)=REG#;                                              | 6842 DECLARATION
            QOPND(INDX)=SYMBOL;                                           | 6858 DECLARATION
            INDX=INDX+1;                                                  | 6874 DECLARATION
            END;  /* OF ELSE DO */                                        | 6886 DECLARATION
         NVAL_OF_QID=IDENT_QNAME(QUADNAME(QNUM+2));                       | 6886 DECLARATION
         NQGRP=NVAL_OF_QID/10;                                            | 6916 DECLARATION
         IF NQGRP¬=2 THEN                                                 | 6934 DECLARATION
            DO;                                                           | 6960 DECLARATION
            MICRO_LABL_FLD(#OF_LINE)=';------------';                     | 6952 DECLARATION
            MICRO_OPRT_FLD(#OF_LINE)='-------------------------;';        | 6968 DECLARATION
            #OF_LINE=#OF_LINE+1;                                          | 6984 DECLARATION
            END;  /* OF IF THEN DO */                                     | 6996 DECLARATION
         END;  /* OF ELSE DO */                                           | 6996 DECLARATION
      END;  /* OF IF THEN DO */                                           | 6996 DECLARATION
END DECLARATION;                                                          | 6996 DECLARATION
/**********************************************************************/ | 7002
/*                        PROCEDURE BLANK                              */ | 7002
/**********************************************************************/ | 7002
BLANK:PROCEDURE(STRING,WIDTH) CHARACTER;                                  | 7002
  DECLARE (WIDTH,L) FIXED, STRING CHARACTER;                              | 7002 BLANK
    L=LENGTH(STRING);                                                     | 7014 BLANK
    IF L>=WIDTH THEN RETURN STRING;                                       | 7040 BLANK
       ELSE RETURN STRING||SUBSTR('              ',0,WIDTH-L);            | 7066 BLANK
END BLANK;                                                                | 7122 BLANK
/**********************************************************************/ | 7128
/*                        PROCEDURE INPUT_QUADS                        */ | 7128
/**********************************************************************/ | 7128
INPUT_QUADS:PROCEDURE;                                                    | 7128
  DECLARE QRECORD CHARACTER;                                              | 7128 INPUT_QUADS
    OUTPUT='INPUT QUADS:';                                                | 7140 INPUT_QUADS
    OUTPUT,QRECORD=INPUT;                                                 | 7160 INPUT_QUADS
    DO WHILE (LENGTH(QRECORD)>0)&(#OF_QUADS<=99);                         | 7242 INPUT_QUADS
       /* MAXIMUM OF INPUT QUADS IS 99 */                                 | 7322 INPUT_QUADS C18 = 99.
       QUADNAME(#OF_QUADS)=SUBSTR(QRECORD,15,4);                          | 7322 INPUT_QUADS
       QUADOPD1(#OF_QUADS)=SUBSTR(QRECORD,26,11);                         | 7346 INPUT_QUADS C19 = 50331648.
       QUADOPD2(#OF_QUADS)=SUBSTR(QRECORD,39,11);                         | 7370 INPUT_QUADS C20 = 167772160.
       QUADOPD3(#OF_QUADS)=SUBSTR(QRECORD,52,11);                         | 7394 INPUT_QUADS
       #OF_QUADS=#OF_QUADS+1;                                             | 7418 INPUT_QUADS
       OUTPUT,QRECORD=INPUT;                                              | 7430 INPUT_QUADS
    END;  /* OF DO WHILE */                                               | 7512 INPUT_QUADS
END INPUT_QUADS;                                                          | 7520 INPUT_QUADS
/**********************************************************************/ | 7526
/*                        PROCEDURE DRIVER                             */ | 7526
/**********************************************************************/ | 7526
DRIVER:PROCEDURE(SQUADS#);                                                | 7526
  DECLARE (SQUADS#,QUAD#_IN_GROUP,QUAD_GROUP) BIT(8);                     | 7526 DRIVER
    DO QNUM=0 TO SQUADS#-1;                                               | 7538 DRIVER
    IF QBRFLG(QNUM)=1 THEN CALL FBRLABL_GEN(QNUM);                        | 7588 DRIVER
    IF SKIP_NEXT_QUAD=0 THEN                                              | 7630 DRIVER
       DO;                                                                | 7656 DRIVER
       NVAL_OF_QID=IDENT_QNAME(QUADNAME(QNUM));                           | 7648 DRIVER
       QUAD#_IN_GROUP=NVAL_OF_QID MOD 10;                                 | 7674 DRIVER
       QUAD_GROUP=NVAL_OF_QID/10;                                         | 7694 DRIVER
       DO CASE QUAD_GROUP;                                                | 7712 DRIVER
          CALL DIR_BRNCH(QUAD#_IN_GROUP);                                 | 7738 DRIVER
          OUTPUT='NO INDIRECT BRANCH HAS BEEN IMPLEMENTED YET';           | 7756 DRIVER CASE 0.
          CALL DECLARATION(QUAD#_IN_GROUP);                               | 7784 DRIVER CASE 1.
          CALL ARITH_OP(QUAD#_IN_GROUP);                                  | 7810 DRIVER CASE 2.
          CALL SHIFT_OP(QUAD#_IN_GROUP);                                  | 7832 DRIVER CASE 3.
          CALL LOGIC_OP(QUAD#_IN_GROUP);                                  | 7858 DRIVER CASE 4.
          CALL MOV_CONV(QUAD#_IN_GROUP);                                  | 7884 DRIVER CASE 5.
          CALL MISC(QUAD#_IN_GROUP);                                      | 7906 DRIVER CASE 6.
       END;  /* OF DO CASE */                                             | 7932 DRIVER CASE 7.
       END;  /* OF IF THEN DO */                                          | 7940 DRIVER
                                                                          | 7940 DRIVER
```

```
END DRIVER;                                                                  |  7962 DRIVER
/************************************************************************/|  7968
                        PROCEDURE OUTPUT_MICRO                           */|  7968
/************************************************************************/|  7968
OUTPUT_MICRO:PROCEDURE;                                                      |  7968 OUTPUT_MICRO
  DECLARE (S1,S2) CHARACTER, N FIXED;                                        |  7980 OUTPUT_MICRO
    DO N=0 TO #OF_LINE;                                                      |  8022 OUTPUT_MICRO
    S1=BLANK(MICRO_LABL_FLD(N),13);                                          |  8058 OUTPUT_MICRO
    S2=BLANK(MICRO_OPRT_FLD(N),13);                                          |  8094 OUTPUT_MICRO
    OUTPUT='          '||S1||S2;                                             |  8148 OUTPUT_MICRO
    END;  /* OF DO STATEMENT */                                              |  8156 OUTPUT_MICRO
END OUTPUT_MICRO;                                                            |  8162
/************************************************************************/|  8162
/*                       ***MAIN PROGRAM***                              */|  8162
/************************************************************************/|  8162
   ALPHA='ABCDEFGHIJKLMNOPQRSTUVWXYZ';                                       |  8170
   DIGIT='0123456789';                                                       |  8178
   CALL INPUT_QUADS;                                                         |  8186
   CALL DRIVER(#OF_QUADS);                                                   |  8202
   OUTPUT(1)='1';                                                            |  8226
   CALL OUTPUT_MICRO;                                                        |  8234
 EOF EOF EOF
```

```
* FILE CONTROL BLOCK  13000    13000    1    1    13000    8244    5968
* LOAD FILE WRITTEN.
END OF COMPILATION SEPTEMBER 2, 1980.  CLOCK TIME = 10:25:46.47.

474 CARDS CONTAINING 260 STATEMENTS WERE COMPILED.
NO ERRORS WERE DETECTED.
8242 BYTES OF PROGRAM, 2133 OF DATA, 2356 OF DESCRIPTORS, 1476 OF STRINGS.  TOTAL CORE REQUIREMENT 14207 BYTES.


SYMBOL  TABLE  DUMP


#OF_LINE          : FIXED     AT 1340(11),   DECLARED ON LINE 7 AND REFERENCED 57 TIMES.
#OF_QUADS         : FIXED     AT 1352(11),   DECLARED ON LINE 8 AND REFERENCED 8 TIMES.
ALPHA             : CHARACTER AT 1304(13),   DECLARED ON LINE 15 AND REFERENCED 5 TIMES.
ARITH_OP          : LABEL     AT 3350(14),   DECLARED ON LINE 171 AND REFERENCED 1 TIMES.
  PARAMETER  1    : BIT(8)    AT 1676(11),   DECLARED ON LINE 172 AND REFERENCED 2 TIMES.
BLANK             : CHARACTER PROCEDURE AT 7010(14),   DECLARED ON LINE 405 AND REFERENCED 2 TIMES.
  PARAMETER  1    : CHARACTER AT 2312(13),   DECLARED ON LINE 406 AND REFERENCED 3 TIMES.
  PARAMETER  2    : FIXED     AT 2040(11),   DECLARED ON LINE 406 AND REFERENCED 2 TIMES.
COMENT_LINE       : LABEL     AT 2062(14),   DECLARED ON LINE 83 AND REFERENCED 12 TIMES.
CONST_ARRAY       : CHARACTER AT 1624(13),   DECLARED ON LINE 29 AND REFERENCED 1 TIMES.
CONVERT_TO_INTEG: LABEL       AT 2116(14),   DECLARED ON LINE 91 AND REFERENCED 1 TIMES.
  PARAMETER  1    : CHARACTER AT 1888(13),   DECLARED ON LINE 92 AND REFERENCED 2 TIMES.
DECLARATION       : LABEL     AT 6404(14),   DECLARED ON LINE 361 AND REFERENCED 1 TIMES.
  PARAMETER  1    : BIT(8)    AT 2010(11),   DECLARED ON LINE 362 AND REFERENCED 0 TIMES.
DIGIT             : CHARACTER AT 1308(13),   DECLARED ON LINE 15 AND REFERENCED 2 TIMES.
DIR_BRNCH         : LABEL     AT 5398(14),   DECLARED ON LINE 276 AND REFERENCED 1 TIMES.
  PARAMETER  1    : BIT(8)    AT 1892(11),   DECLARED ON LINE 277 AND REFERENCED 1 TIMES.
DRIVER            : LABEL     AT 7534(14),   DECLARED ON LINE 431 AND REFERENCED 1 TIMES.
  PARAMETER  1    : BIT(8)    AT 2076(11),   DECLARED ON LINE 432 AND REFERENCED 1 TIMES.
FBRDESQ           : BIT(8)    AT 1392(11),   DECLARED ON LINE 11 AND REFERENCED 2 TIMES.
FBRLABL_GEN       : LABEL     AT 5996(14),   DECLARED ON LINE 326 AND REFERENCED 1 TIMES.
  PARAMETER  1    : BIT(8)    AT 1972(11),   DECLARED ON LINE 327 AND REFERENCED 1 TIMES.
FIND_REG#         : CHARACTER PROCEDURE AT 2240(14),   DECLARED ON LINE 102 AND REFERENCED 9 TIMES.
  PARAMETER  1    : CHARACTER AT 1896(13),   DECLARED ON LINE 103 AND REFERENCED 1 TIMES.
GLABL             : LABEL     AT 4390(14),   DECLARED ON LINE 216 AND REFERENCED 2 TIMES.
  PARAMETER  1    : CHARACTER AT 2092(13),   DECLARED ON LINE 217 AND REFERENCED 1 TIMES.
IDENT_CONST       : LABEL     AT 1494(14),   DECLARED ON LINE 52 AND REFERENCED 2 TIMES.
  PARAMETER  1    : CHARACTER AT 1860(13),   DECLARED ON LINE 53 AND REFERENCED 1 TIMES.
IDENT_QNAME       : LABEL     AT 1290(14),   DECLARED ON LINE 38 AND REFERENCED 4 TIMES.
  PARAMETER  1    : CHARACTER AT 1852(13),   DECLARED ON LINE 39 AND REFERENCED 1 TIMES.
IN#               : FIXED     AT 1344(11),   DECLARED ON LINE 7 AND REFERENCED 4 TIMES.
INDUM             : BIT(8)    AT 1436(11),   DECLARED ON LINE 13 AND REFERENCED 5 TIMES.
INDX              : FIXED     AT 1348(11),   DECLARED ON LINE 8 AND REFERENCED 5 TIMES.
INPUT_QUADS       : LABEL     AT 7136(14),   DECLARED ON LINE 414 AND REFERENCED 1 TIMES.
LABL_FLD_INDX     : BIT(8)    AT 1358(11),   DECLARED ON LINE 10 AND REFERENCED 8 TIMES.
LOGIC_OP          : LABEL     AT 6366(14),   DECLARED ON LINE 354 AND REFERENCED 1 TIMES.
  PARAMETER  1    : BIT(8)    AT 2000(11),   DECLARED ON LINE 355 AND REFERENCED 0 TIMES.
MEM_LABL_GEN      : LABEL     AT 6290(14),   DECLARED ON LINE 341 AND REFERENCED 1 TIMES.
MICRO_LABL_FLD    : CHARACTER AT 568(13),    DECLARED ON LINE 5 AND REFERENCED 9 TIMES.
MICRO_OPRT_FLD    : CHARACTER AT 852(13),    DECLARED ON LINE 5 AND REFERENCED 27 TIMES.
MISC              : LABEL     AT 5754(14),   DECLARED ON LINE 310 AND REFERENCED 1 TIMES.
  PARAMETER  1    : FIXED     AT 1932(11),   DECLARED ON LINE 311 AND REFERENCED 1 TIMES.
MOV_CONV          : LABEL     AT 3274(14),   DECLARED ON LINE 160 AND REFERENCED 1 TIMES.
  PARAMETER  1    : BIT(8)    AT 1656(11),   DECLARED ON LINE 161 AND REFERENCED 1 TIMES.
NVAL_OF_QID       : BIT(8)    AT 1357(11),   DECLARED ON LINE 9 AND REFERENCED 8 TIMES.
OPND              : CHARACTER AT 1136(13),   DECLARED ON LINE 6 AND REFERENCED 2 TIMES.
OUTPUT_MICRO      : LABEL     AT 7976(14),   DECLARED ON LINE 457 AND REFERENCED 1 TIMES.
PROC_3G           : LABEL     AT 4980(14),   DECLARED ON LINE 251 AND REFERENCED 1 TIMES.
```

```
PROC_MOV          : LABEL     AT 2428(14),   DECLARED ON LINE 119 AND REFERENCED 1 TIMES.
PROC_RETN         : LABEL     AT 5716(14),   DECLARED ON LINE 304 AND REFERENCED 1 TIMES.
PROC_UNS          : LABEL     AT 5558(14),   DECLARED ON LINE 289 AN   FFERENCED 1 TIMES.
QBKFLG            : BIT(8)    AT 1402(11),   DECLARED ON LINE 12 AND  EFERENCED 2 TIMES.
QNUM              : BIT(8)    AT 1356(11),   DECLARED ON LINE 9 AND REFERENCED 23 TIMES.
QOPND             : CHARACTER AT 1220(13),   DECLARED ON LINE 6 AND REFERENCED 2 TIMES.
QUAD_TAB          : CHARACTER AT 1312(13),   DECLARED ON LINE 20 AND REFERENCED 1 TIMES.
QUADNAME          : CHARACTER AT 24(13),     DECLARED ON LINE 4 AND REFERENCED 5 TIMES.
QUADOPD1          : CHARACTER AT 160(13),    DECLARED ON LINE 4 AND REFERENCED 6 TIMES.
QUADOPD2          : CHARACTER AT 296(13),    DECLARED ON LINE 4 AND REFERENCED 3 TIMES.
QUADOPD3          : CHARACTER AT 432(13),    DECLARED ON LINE 4 AND REFERENCED 4 TIMES.
SHIFT_OP          : LABEL     AT 6328(14),   DECLARED ON LINE 347 AND REFERENCED 1 TIMES.
  PARAMETER  1    : BIT(8)    AT 1992(11),   DECLARED ON LINE 348 AND REFERENCED 0 TIMES.
SKIP_NEXT_QUAD    : BIT(8)    AT 1437(11),   DECLARED ON LINE 14 AND REFERENCED 4 TIMES.
SYMBTAYL          : CHARACTER PROCEDURE AT 1660(14),   DECLARED ON LINE 65 AND REFERENCED 10 TIMES.
  PARAMETER  1    : CHARACTER AT 1864(13),   DECLARED ON LINE 66 AND REFERENCED 8 TIMES.
```

```
MACRO DEFINITIONS:

DX_SIZE              LITERALLY: 500

IDCOMPARES          = 16966
SYMBOL TABLE SIZE = 99
MACRO DEFINITIONS = 1
STACKING DECISIONS= 11886
SCAN                = 3316
EMITRR              = 361
EMITRX              = 1977
FORCEACCUMULATOR   = 778
ARITHEMIT           = 133
GENSTORE            = 188
FIXBFW              = 149
FIXDATAWORD         = 12
FIXCHW              = 209
GETDATA             = 0
GETCODE             = 0
FINDADDRESS         = 472
SHORTCFIX           = 209
LONGCFIX            = 0
SHORTDFIX           = 12
LONGDFIX            = 0
FREE STRING AREA   = 111222

REGISTER VALUES (RELATIVE TO R11):
R4 = 0
R5 = 0
R6 = 0
R7 = 0
R8 = 0
R9 = 0
R10 = 0
R11 = 0
R12 = 0
R13 = 2128

 INSTRUCTION FREQUENCIES:

BALR     36
BCTR     3
BCR      40
LPR      1
LTR      8
NR       12
OR       4
XR       17
LR       53
CR       7
AR       22
SR       157
DR       1
STH      3
LA       156
STC      56
IC       99
```

```
BC        224
LH
ST          3
N         18
O         23
L         529
C         48
A         42
S         19
M         1
D         5
AL        2
SL        4
SRL       22
SLL       73
SRA       17
SROA      5
STM       36
TM        1
DI        1
LM        37

TOTAL TIME IN COMPILER    0:2:30.53.
SET UP TIME               0:0:14.65.
ACTUAL COMPILATION TIME   0:1:44.17.
POST-COMPILATION TIME     0:0:31.71.
COMPILATION RATE: 273 CARDS PER MINUTE.
```

```
INPUT QUADS:
     0  | ADDR    ,  U000000010,              ,  0000000000  |
     1  | ADDR    ,  U000000011,              ,  0000000002  |
     2  | BYTE    ,             ,  0000000001,              |
     3  | ADDR    ,  U000000012,              ,  0000000004  |
     4  | BYTE    ,             ,  0000000001,              |
     5  | ADDR    ,  U000000013,              ,  0000000006  |
     6  | BYTE    ,             ,  0000000001,              |
     7  | ADDR    ,  U000000314,              ,  0000000008  |
     8  | BYTE    ,             ,  0000000001,              |
     9  | ADDR    ,  U000000015,              ,  0000000010  |
    10  | BYTE    ,             ,  0000000001,              |
    11  | ADDR    ,  U000000016,              ,  0000000012  |
    12  | BYTE    ,             ,  0000000001,              |
    13  | MOV     , #0000000000,              ,  U000000012  |
    14  | MOV     , #0000000001,              ,  U000000013  |
    15  | MOV     , #0000000010,              ,  U000000014  |
    16  | MOV    I, #0000000001,              ,  U000000015  |
    17  | BG      ,  0000000024,  U000000015,  U000000014  |
    18  | MOV     ,  U000000012,              ,  U000000011  |
    19  | MOV     ,  U000000013,              ,  U000000012  |
    20  | ADD    I,  U000000011,  U000000012,  U000000016  |
    21  | MOV     ,  U000000016,              ,  U000000013  |
    22  | ADD    I, #0000000001,              ,  U000000015  |
    23  | BR      ,  0000000017,              ,              |
    24  | UNST    ,             ,              , #0000000004  |
    25  | RETN    ,             ,              ,              |
```

```
                .TOC '
                .REGION   /1600,17FF
;INPUT
;               R0 -- U11
;               R1 -- U12
;               R2 -- U13
;               R3 -- U14
;               R4 -- U15
;               R5 -- U16
;----------------------------------------;
                R(.R1.)_K(..ZERO.);
                ;------------------------;
                R(.R2.)_K(..1.);
                ;------------------------;
                R(.R3.)_K(..10.);
                ;------------------------;
                R(.R4.)_K(..1.);
                ;------------------------;
LABL-A:         Q_R(.R4.);
                ;------------------------;
                ALU_LA-Q,
                LAB_R(.R3.),
                CLK.UBCC;
                ;------------------------;
                ALU?;
=1110           ;------------------------;
                J/LBLF-A;
                ;------------------------;
                LAB_R(.R1.);
                ;------------------------;
                R(.R0.)_LA;
                ;------------------------;
                LAB_R(.R2.);
                ;------------------------;
                R(.R1.)_LA;
                ;------------------------;
                Q_R(.R0.);
                ;------------------------;
                LAB_R(.R1.);
                ;------------------------;
                R(.R5.)_ALU, ALU_LA+Q;
                ;------------------------;
                LAB_R(.R5.);
                ;------------------------;
                R(.R2.)_LA;
                ;------------------------;
                LAB_R(.R4.);
                ;------------------------;
                ALU_LA+K(..1.),
                R(.R4.)_ALU;
                ;------------------------;
                J/LABL-A;
                ;------------------------;
LBLF-A:         PC_PC+1, CLR.IB.DPC,
                J/062;
                ;------------------------;
```

APPENDIX B

"USER" MICROPROGRAMMING MANUAL FOR THE DEC VAX 11/780

Teh-Hsin Yang

CONTENTS

1) <u>Basic Architecture Features of VAX 11/780</u>

The VAX 11/780 is a general purpose microprogrammed computer. It features a 32 bit word, virtual sotrage which provides addressing of 2B Bytes, an 8K cache storage unit, a 200 NS basic CPU cycle, and a powerful instruction set. This system includes 1024 words(96 bits) of "user" microprogrammable control store. Support software for microprogramming includes an assembler and debugger which permits the user to design their own microcode. It has been demonstrated that an algorithm expressed directly in microcode will execute faster than routines programmed in assembly or a high level language (HLL). Before describing this procedure for preparing microprograms for the VAX 11/780, a brief overview of the architecture of the CPU will be given. Special attention will be given to those parts which most profoundly affect the preparation of microprograms. Figure D1, which is a block diagram of the VAX 11/780 CPU, will support the ensuing discussion.

The VAX 11/780 CPU has two sets of sixteen 32 bit general purpose registers. A number of these registers have preassigned functions (R_{12} = Argument list pointer, R_{13} = Stack frame pointer, R_{14} = Stack pointer, R_{15} = PC. Registers R0 thru R7 are available to programmers. The output of the A registers passes through latch LA to the A side of the ALU while the output of the B registers passes through latch LB to the B side of the ALU. An additional set of 16 working 32 Bit Registers, RC, output thru an LC latch to the left side of the ALU. The RC registers are only available at the microcode level.

Two other 32 bit registers are named D and Q. They can be gated to either side of the ALU. The content of D register can be delivered to the Q Register directly without passing through the ALU. A rotation unit accepts the 64 bits from the Q and D registers and rotates it by the amount specified by a user instruction and deposits 32 bits of the shifted result back in the D register. The D register also acts as the memory data register and data read from and into storage must pass through this register.

The memory address register, VA, can be loaded from ALU or can be incremented by 4 which advances the address by one word (32 bits). There

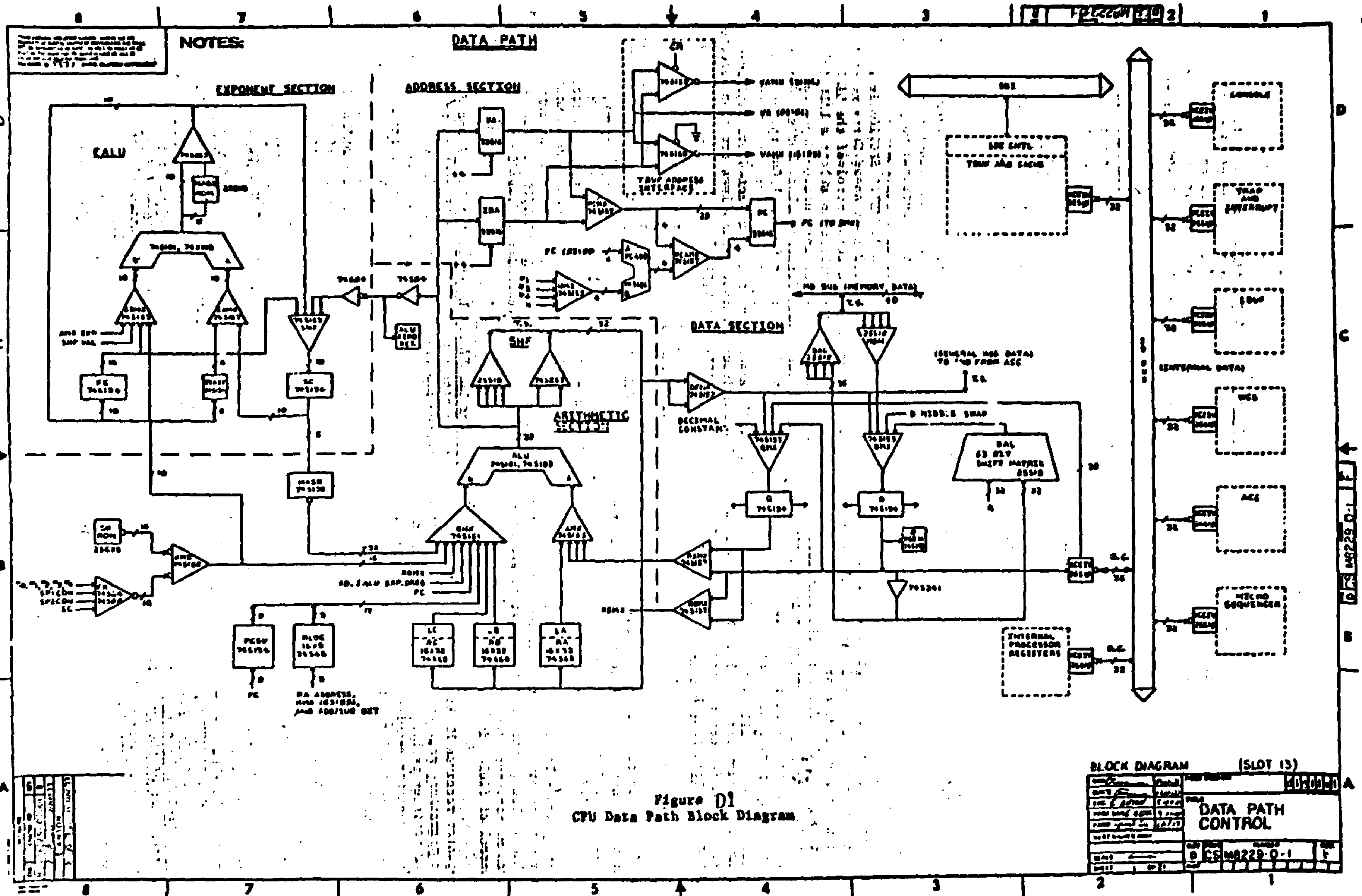


Figure D1
CPU Data Path Block Diagram

is an auxiliary 10-bit exponent ALU (EALU) for handling floating point exponent computations. Its data path contains a register named SC which is used for shift operations and masking the data entering the left side of the ALU. Another feature of the VAX CPU is the constant generator. A ROM, called SK, provides 64 16 bit constants available to the left side of ALU.

The VAX machine is controlled by a horizontal micro control word which is 96 bits wide and divided into 30 control fields. This represents a horizontal control word format and presents a complicated microprogram design problem. Many operations may be carried out in parallel. A detailed description of the micro control word is not given here since it is described in other manuals. But it will be useful to give a brief overview of branch micro-operations for each control word. Thirteen bits are used to form the address of next micro instruction. These can be assigned directly by the microprogram which corresponds to an unconditional branch at the microcode level. When a conditional branch is required, other micro control words are involved in establishing the branch conditions required. The BEN (Branch Enable) micro control field contains 5 bits which are used to specify twenty-six branch conditions. From this control field it is possible to select different program status words (PSWs), including the N, Z, V, C and other flags as Branch conditions. These conditions are ORed with the low-order bits of the micro address field (JMP) to form the address of the successor microword. In order to make a conditional branch occur, it is necessary to make sure the potential branch address has certain bits equal to zero so that the OR operation produces the appropriate branch address. The Micro-2 Assembler, to be described later, has the ability to make appropriate bits of a branch address equal to zero to satisfy the OR requirements.

2) <u>Micro 2 - Microprogram Assembler</u>

The process of writing, loading, and executing a microprogram in the VAX 11/780 "User" control storage area is described here. Figure D2 is a Flowchart which shows these steps. The first step in microprogramming the VAX 11/780 is generation of the microcode source load module. To do this, use is made of the VAX SOS Editor system which is initiated thru use of the edit command. The individual microprogram statements are entered along with comments as shown by example in Figure D3. While not required, certain formating procedures are recommended to improve readability of this microcode. The formating procedures are:

MICROCODE GENERATION PROTOCALS

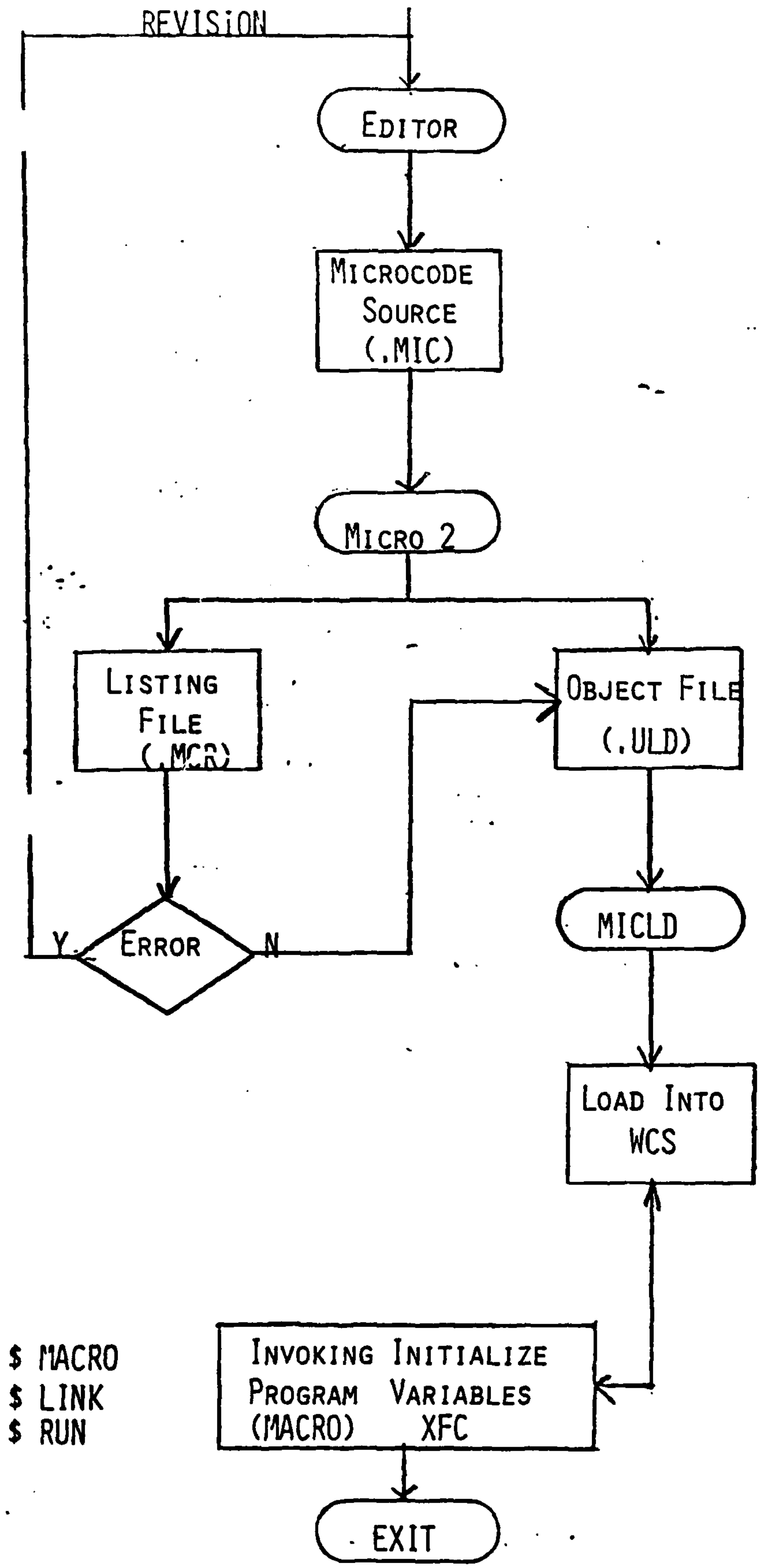
1. Precede Micro instruction by any general comments

2. If Micro instruction has explicit address--give that address at the left margin--put no info on that line.

3. If Micro instruction has a label--give label at left margin and put no other information on that line.

4. Include as many Micro instruction--parts separated by commas as will fit in columns starting at the second tab (col 17) and going to column 38.

5. Place line specific comments starting at 5th tab Col 41.



COMMAND

$EDIT FILE NAME .MIC

$MICRO 2 FILE NAME

$MICLD FILE NAME

FIGURE D2 GENERATION OF MICROCODE FOR VAX 11/780

```
TYPE BUBBLE.MIC;8
                        .TITLE   'BUBBLE  SORT'
                        .TOC  'ATTACHED MACRO DEFINITIONS'
  ALU_Q+LB-1            'RAMX/Q,AMX/RAMX,BMX/LB,ALU/A+B'
                        .REGION   /1400,17FF
;   INPUTS
;                       ASSUMPTION
;                       R1-I
;                       R2-J
;                       R3-K
;                       R4-TEMPORARY STORAGE
;                       R6-N
;                       R7-START ADDRESS  OF ARRAY
;-------------------------------------------;
                        Q_R[R7]          ; ARRAY START ADDRESS-Q
;-------------------------------------------;

START:                  ALU_LA-K[.1],
                        LA_RA[R3]        ; K-1 TO K
;-------------------------------------------;
                        R[R3]_ALU        ;
;-------------------------------------------;
                        R[R1]_ALU,
                        ALU_K[.1]        ; I=1
;-------------------------------------------;
LOOP:                   D_LA,
                        LA_RA[R3]        ; K--D
;-------------------------------------------;
                        ALU_D-LB,
                        LAB_R[R1],CLK.UBCC ;K-I IN ALU
;-------------------------------------------;
                        ALU.N?           ; K>I?
;-------------------------------------------;
=0111                                    ;K>I
                        D_ALU.LEFT2,
                        ALU_R[R1]        ; I*4  TO  D
                        VA_ALU,
                        ALU_Q[+]D,J/PP   ;GET ARRAY(I) ADDRES
;-------------------------------------------;
                                         ;  K<I
                        ALU_LA-K[.6],
                        LA_RA[R3],CLK.UBCC;K-0
;-------------------------------------------;
                        C31?             ;  K>0?
;-------------------------------------------;
=0                                       ;  C31=0 K>0
                        J/START          ;  BACK TO START
;-------------------------------------------;
                                         ;  C31=1 K<=0
                        PC_PC+1,CLR.IB.OPC,J/062  ;  STOP
;-------------------------------------------;
PP:                     D[LONG]_CACHE    ;  READ FROM MEMORY
;-------------------------------------------;
                        R[R4]_D;
;-------------------------------------------;
                        LAB_R[R4]   ;   ARRAY(I)     TO LA
;-------------------------------------------;
                        VA_VA+4,LAB_R[R4]; GET ARRAY(I+1) ADDR

;-------------------------------------------;
                        D[LONG]_CACHE    ;  READ ARRAY(I+1) TO D
;-------------------------------------------;
                        ALU_LA-D,CLK.UBCC ; ARRAY(I)-ARRAY(I+1)
;-------------------------------------------;
                        LAB_R[R1],ALU.N?  ;I TO LB
;-------------------------------------------;
=0111     ;ALU.N=0,ARRAY(I)>ARRAY(I+1)

;-------------------------------------------;
          ;ALU.N=1,ARRAY(I+1)<=ARRAY(I)
                        VA_ALU,
                        ALU_Q+LB         ;ARRAY(I+1) ADDRESS TO VA
;-------------------------------------------;
                        CACHE_D[LONG],
                        ALU_Q+LB-1       ;WRITE (ARRAY(I)) TO ARRAY(I+1)
;-------------------------------------------;
                        VA_ALU           ;ARRAY(I) ADDRESS TO VA
;-------------------------------------------;
                        D_R[R4] ;ARRAY(I+1)   TO D
;-------------------------------------------;
                        CACHE_D[LONG],
                        J/LOOP           ;WRITE (ARRAY(I+1)) TO ARRAY(I)
;-------------------------------------------;
                                         ;END BUBBLE SORT
```

Figure 3- Microcode Source Program for MICRO 2 Assembler

Initially a source file name followed by .MIC is entered which notifies the VAX/VMS operating system that the editor is creating source microcode modules. By using the command $MICRO 2, "file name", the operating system is notified that the MICRO 2 assembler is to be used to create a microcode load module which will be identified as .MCR and will be file .MCR; N where N is the number of Micro Assemblies executed. The MICRO 2 language lets the microprogrammers express the actions of a microprogram symbolically. The MICRO 2 assembler translates this symbolic representation into micro control words. MICRO 2 also allocates control storage space to any microwords for which it isn't specifically identified. In allocating control storage address, the microprogrammer has three choices:

--Allocating control storage location for a microword absolutely.

--Specifying a constraint on its control stoarge allocation of a microword e.g. making the lowest order bits of the address equal zero.

--Letting MICRO 2 determine the microword allocation.

MICRO 2 detects syntax errors in the source microprogram. The syntax checking is based on two parts:

--First, it checks whether the statements are recognizable by MICRO 2. Unrecognizable statements create an error message.

--Second, a check of data path conflicts is made. If the data paths overlap for a particular microinstruction, an error message is generated.

The MICRO 2 source language also contains some "pseudo-operations" which don't generate microcode but give commands to the assembler. This is done by either specifying a command to the assembler, or by influencing the output of the assembler. In order to simplify the writing of a microprogram, the VAX 11/780 "user" microprogram support offers a group of MACRO definitions expressed in a language recognized by MICRO 2. These MACRO definitions are presented in the VAX 11/780 microprogramming tools user's guide.

They are categorized as described below:

.ALU_0 . . . thru ALU_D. AND . . .

ALU_0(B). . thru ALU_D.XOR

```
;ALU_K . . . thru ALU_PC . . .
;ALU_Q . . . thru CACHE_ . . .
;D_0   . . . thru D_CACHE
;D_DAL . . . thru D_D . . .
;D_INT.SUM . . .thru D_PC . . .
;D_Q . . .
;D_R . . . thru D & VA_ . . .
;EALU_ . . . thru FE_ . . .
;ID_ . . . thru LC_. . .
;PC_ . . . thru PC & VA_  . . .
;Q_0 . . . thru Q_D  . . .
;Q_IB  . . . thru Q_PC
;Q_Q . . . thru Q & VA_. . .
;R[  ]_0 . . . thru R[  ] _PACK.FP
;RC[  ] _0 . . . thru RC [  ] _D . . .
;SC_0(A)
;SD_ . . . thru VA_ . . .
;MATCH States
;Non transfer functions
;Branch enable Macro definitions
```

If the microprogrammer doesn't find a MACRO which satisfies a requirement, two steps can be taken. One is to specify the data path and microoperations directly which is time consuming and difficult. The second is to define a new Macro which accomplishes the desired CPU action.

3) How to load and execute a microprogram.

Upon completion of the generation of a source microprogram in the MICRO 2 assembly language including corrections to all errors detected, the next step is to load the microprogram into "user" control storage. This is accomplished by first assembling the source microprogram by issuing the command:

$MICRO 2 file name and depressing the return key.

The MICRO 2 assembler assembles the source program and produces a listing file (.MCR). To obtain a copy of the assembled program, issue the command:

$TYPE or PRINT file name .MCR and depress the return key.

Depending on the type of terminal being used the response will be a listing either on the CRT display, a keyboard printer (TYPE) or on a line printer (PRINT). The listing shows the source program and the corresponding microcode. It also indicates syntax errors if any exist. When all errors are corrected, use the command:

$MICLD

The assembled program is loaded into writable control storage (WCS). The accomplishment of this load is specified at a CRT terminal by the appearance of the prompt symbol indicating that the VMS control program is awaiting the next command. In order to execute the assembled microprogram it is necessary to transfer control of the VAX 11/780 CPU over to the microcode now resident in the WCS. This requires a command to the Virtual Machine System (VMS). To do this an invoking program written in VAX MACRO assembler language is required. The VAX 11/780 MACRO assembly language has an extended function call, (XFC), instruction which transfers control from VMS to the microcode in WCS. The invoking program must also set up any parameters to be passed to the microcode either by inserting values in registers or through a parameter list in main memory. A pointer to this parameter list must be stored in a register. Also to be able to check the result of running a microprogram a labeled NOP instruction serves as a breakpoint which can be used to check the result of executing microcode by using the MACRO Debug facility. The sample invoking program which in addition to executing the microcode in WCS also measures microcode execution time is shown in figure D5.

```
  TYPE  SORTTEST.MAR
.TITLE          SORTTEST
.ENTRY          SORTTEST,^M<>
$CMKRNL_S       ROUTIN=SAVEC     ;SAVE AND CHANGE XFC VECTOR IN SCB.
$CREATE         FAB=OUTFAB
$CONNECT        RAB=OUTRAB
$ASCTIM_S       ,TIMBUF=ATIMENOW,,
MOVC3           #32,ATIMENOW,TIMOUT
$PUT            RAB=OUTRAB
                MOVAL   AR,R0    ;STARTING ADDR OF ARRAY
;---------------;
XFC
;---------------;
$CMKRNL_S       ROUTIN=RSVEC
$EXIT_S
$ASCTIM_S       ,TIMBUF=ATIMENOW,,
MOVC3           #32,ATIMENOW,TIMOUT
$PUT            RAB=OUTRAB
$CLOSE          FAB=OUTFAB
;---------------;
SAVEC:          .WORD   0
                MOVL    SCB$AL_BASE+20,RIVEC
                MOVL    #2,SCB$AL_BASE+20
                RET
;---------------;
RSVEC:          .WORD   0
                MOVL    RIVEC,SCB$AL_BASE+20
                RET
;---------------;
                .PSECT  VECTOR,LONG
RIVEC:          .LONG   0
                .LONG   0
;---------------;
OUTFAB:         $FAB    FNM=TIMBF,RFM=FIX,MRS=TIMSIZ,RAT=CR
OUTRAB:         $RAB    FAB=OUTFAB,RBF=TIMOUT,RSZ=TIMSIZ
TIMOUT:         .BLKB   32
                TIMSIZ=.-TIMOUT
ATIMENOW:       .LONG   20$-10$
                .LONG 10$
10$:            .BLKB   32
20$:            .BLKB   0
AR:             .BLKL   255
                .END    SORTTEST
$
```

Figure 5 Microcode Invoking Program in VAX 11/780 MACRO Language

The steps to use the VMS commands to execute the MACRO invoking program assuming the microprogram is loaded in WCS are:

```
$ MACRO  invoking file name
$ LINK   invoking file name + SYS $ System:SYS.STB/SEL
$ RUN    invoking file name
```

Execution begins under VMS of the MACRO invoking program and when the XFC instruction is encountered, the microcode is executed. Control is returned to VMS and the balance of the assembly language program is executed.

4) Control Command to Debug the "User" Microprogram.

It is Very important to have a debug facility to check the MICRO Code when it is running in WCS. MICRO 2 includes commands that can cause the microprogram to be executed step by step or to execute up to any microinstruction in the microprograms in WCS. A list of user's commands and terminal reaction is shown below:

USER	COMMAND	TERMINAL INFORMATION	COMMENTS
Prompt	Command		
$	MACRO File Name	Nothing, if nothing is wrong	Assemble the invoking program
$	LINK File Name SYSTEM:SYS. STB/SEL	Nothing, if nothing is wrong	Link the invoking program
$	MICRO 2 File Name	Nothing, if nothing is wrong	Assemble the microcode
$	MICLD File Name	WCS Load Complete	Loads microcode into WCS
$	CTRL	Show >>>	Turn to microprogram
$	RUN File Name	>>> WCS	Start Execution of invoking Program & invokes WCS Debugger
>>>	SE SOMM	Show >>>	Set-Stop on Micro Match SOMM
>>>	H	Halted at XXX	Halt
>>>	C	Nothing	Continue
WCS >	C	Message as appropriate when execution stops	Continue to execute next instruction
WCS >	E RA [ ]	Show content of register RA (HEX)	Exam the content of RA [ ]
WCS >	E DR	Show content of D register	
WCS >	RET	>>> Information on CPU status	Return to MACRO Control
>>>	Show	Shows CPU status	
>>>	E/ID 21	E: Examine ID: Internal Register 21: SOMM Register	

APPENDIX C

QUADRUPLE DEFINITIONS

The following quadruple definitions are to be considered standard for the interface between the PASCAL and XPL to quadruple compilers and the Quadruple to VAX MACRO (Assembly Language) and MICRO 2 (Microcode) translators.

Two formats are described below. One is for the complete quadruple while the second is for operands 1, 2 and 3:

Quadruple Format:	Operation	Data Type	Operand 1	Operand 2	Operand 3
	14 17	22	25 35	38 48	51 61
	xxxxxxxxx	I			
		R			
		b			

Operand Format

Operand 1	25	26	27	35
Operand 2	38	39	40	48
Operand 3	51	52	53	61
	#	U	N	N
	@	T		
		N		

R	Operands are real values
I	Operands are integer values
#	Operand is a numeric value
@	Operand is an address of a data item
$	
T	Operand is temporary location in storage on a register
U	Operand is a storage address assigned to a data item
N	Numeric value

OPERATOR	OPRND 1	OPRND 2	OPRND 3	FUNCTIONAL DEFINITION
MOV	# N UNNNNNNNNN T @		UNNNNNNNNN T @	Move numeric, storage location, Temporary storage location or indirect address value specified in operand 1 to the storage or temporary storage location or as an indirect address in operand 3.
ADD/SUB	# N UNNNNNNNNN T		UNNNNNNNNN T	Add/Sub, storage location, or temporary stoarge location value to/from the numeric, storage location, or temporary storage location value specified in operand 1 and place the storage or temporary storage location of the result and value in operand 3.
ADD/SUB	# N UNNNNNNNNN T	# N UNNNNNNNNN T	UNNNNNNNNN T	Add/Sub numeric, storage location, or temporary storage location value specified in operand 2 to/from numeric, storage location or temporary storage location value specified by operand 1 and place the storage or temporary storage location of the resultant value in operand 3.
SHLA/SHRA	UNNNNNNNNN T	# NNNNNNNNNN	UNNNNNNNNN T	Shift storage location or temporary storage location value specified in Operand 1 by an amount specified by the numeric value specified in operand 2 and place the shifted resultant value in the storage or temporary storage location specified in operand 3.

OPERATOR	OPRND 1	OPRND 2	OPRND 3	FUNCTIONAL DEFINITION
INDX	UNNNNNNNNN T	UNNNNNNNNN T	UNNNNNNNNN T	Add the storage or temporary storage value specified in operand 2 to the storage or temporary storage value specified in operand 1 and place the resultant and value in the storage or temporary storage location specified by operand 3.
LT/GT/ET	# N UNNNNNNNNN T	# N UNNNNNNNNN T	TNNNNNNNNN	Subtract the numeric, storage or temporary storage location value specified in operand 2 from the numeric, storage or temporary storage location value specified in operand 1 and check the sign and zero output flags. For LT and sign + store 1 for True in temporary storage location specified by Oprnd 3. For GT and sign - store 1 for true in temporary storage location specified by Oprnd 3. For ET and zero flag set store 1 for true in temporary storage location specified by Oprnd 3.
BRT/BRF	NNNNNNNNNN	UNNNNNNNNN T		Branch to the storage address specified in operand 1 which is either a label or an absolute address if the value stored in the storage or temporary storage location specified in operand 2 is 1 for true (T) or 0 for False (F).
BR	TNNNNNNNNN U			Branch to the storage address specified in operand 1

OPERATOP	OPRND 1	OPRND 2	OPRND 3	FUNCTIONAL DEFINITION
BE/BG/BL	NNNNNNNNNN	UNNNNNNNNN T	UNNNNNNNNN T	Branch to the storage address specified in operand one if the result of subtracting the storage or temporary storage location value specified in operand 3 from the storage or temporary storage location value specified in operand 2 is = 0 (E); >0 (G); <0 (L);
Byte/Half/Full		NNNNNNNNNN		The number of bytes/half words/ full words/ of storage space to be allocated to a variable specified in a immediately preceding ADDR quadruple is specified by the numeric value in operand 2.
ADDR	UNNNNNNNNN		NNNNNNNNNN	The storage location is specified in operand 1 and the quadruple number of the corresponding Byte/Half/Full quadruple is specified by operand 3.
END				Indicates end of quadruples for a procedure
RET				Indicate a return to a calling procedure
UNST				Indicates that certain variables associated with this procedure should be removed from the compiler stacks.

APPENDIX - D

QUADRUPLES TO MICRO CODE SPECIFICATION

ASSUMPTIONS:

1) The constant which does not exist in Fk, Sk would be stored in Reg C

2) 1 Micro Routine if constant is in Sk

   2 Micro Routine if constant is not in Sk

3) #-Numerical Value

4) S - Storage Address

5) T - Temporary Storage Address

6) S(R) - Storage Address in Register

7) RA [ ] - A Register value
   RB [ ] - B Register value
   RC [ ] - C Register value

8) D [ ] - D Register value
   Q [ ] - Q Register value

9) @R - Indirect address of variable in storage stored in a register
   @T - Indirect address of variable in storage stored in a temporary storage locatio

10) ADDR - Branch Address

QUAD OPERATOR	QUAD OPERAND	MICRO 2
MOV	#, ,S(R)	1 R[x]_K[#]; If the const exist in K 2 LC_RC[ ]; If the const did not exist in K R[RX]_LC ;
MOV	#, ,S	1 D_K[#]; VA_RC[ ]; Cache _ D[ ]; 2 D_RC[ ]; VA_RC[ ]; Cache _ D[ ];
MOV	S, ,S(R)	VA_RC[ ]; D[ ]_Cache; R[ ]_D;
MOV	S, ,S	VA_RC[ ]; D[ ]_Cache; VA_RC[ ]; Cache_D[ ];
MOV	@T_1, ,@T_2	VA_R[T_1]; D[ ]_Cache; VA_R[T_2]; Cache_D[ ];
MOV	@T, ,S	VA_R[T]; D[ ]_Cache; VA_R[T_2]; Cache_D[ ];
MOV	#, ,@T	1 D[ ]_K[#]; VA_R[ ]; Cache_D[ ]; 2 D[ ]_RC[ ]; VA_R[ ]; Cache_D[ ];
INDX	R,# ,T	1 LAB_R[R]; ALU_LA + K[#], R[ ]_ALU; 2 D_RC[ ]; ALU_D + K[#], R[ ]_ALU;
GT	S,# ,T	1 Q_K[#]; ALU_LA_Q, LAB_R[ ], CLK.UBCC; ALU ? ; =1110 J/ADDR;

QUAD OPERATOR	QUAD OPERAND	MICRO 2
		② Q_RC[ ]; ALU_LA_Q, LAB_R[ ], CLK.UBCC; ALU ? ; = 1110 J/ADDR;
GT BRT	S_1 , S_2 , T ADDR, ,	Q_R[S_2]; ALU_LA_Q, LAB_R[S_1], CLK.UBCC; ALU ? ; = 1110 J/ADDR;
ADD	# , S(R), S(R)	① LAB_R[S], ALU_LA + K[ ], R[S]_ALU; ② Q_RC[ ]; LAB_R[S]; ALU_LA + Q, R[S]_ALU;
ADD	S_1(R), S_2(R), S_3	VA_RC[ ], LAB_R[S_1]; LA_R[S_2], ALU_LA[+]LB, D_ALU; Cache_D[Long];
SUB	S(R), # , S(R)	① LAB_R[S]; ALU_LA_K[ ]; R[S]_ALU; ② Q_RC[ ]; LAB_R[S]; ALU_LA_Q, R[S]_ALU;
SUB	S_1(R), S_2(R), S_3	VA_RC[ ], LAB_R[S_2]; LA_R[S_1], ALU_LA[-]LB, D_ALU; Cache_D[Long];
END	, ,	PC_PC + 1, CLR.IB.OPC, J/062;

DAT

FILM

0